Utveckling av webbplats och skräddarsytt innehållshanteringssystem för utbildningsprogrammet Film och Tv vid yrkeshögskolan Arcada

Kristoffer Parikka

Examensarbete

Informationsteknik

2011

EXAMENSARBETE	
Arcada	
Utbildningsprogram:	Informationsteknik
Identifikationsnummer:	3124
Författare:	Kristoffer Parikka
Arbetets namn:	Utveckling av webbplats och skräddarsytt innehållshanteringssystem för utbildningsprogrammet Film och Tv vid yrkeshögskolan Arcada
Handledare (Arcada):	Hanne Karlsson
Uppdragsgivare:	Arcada/utbildningsprogrammet Film och TV

Sammandrag:

Detta examensarbete handlar om utveckling av en webbplats med ett skräddarsytt innehållshanteringssystem gjort för utbildningsprogrammet i Film och TV vid Arcada. Syftet med projektet var att skapa den programvaru-tekniska delen av en webbplats, där utbildningsprogrammet, kan presentera olika inriktningsalternativ och produktioner gjorda av studeranden. Webbplatsen kommer inte att integreras i Arcadas officiella webbportal.

Examensarbetet består av två delar. I den teoretiska delen utreds vad designmodellen MVC, ramverk och innehållshanteringsverktyg är, vad de används till samt vilka huvuddelar det i allmänhet består av. Dessutom undersöks fördelar ett skräddarsytt innehållshanteringssystem kan ge. Sedan följer en detaljerad presentation av ramverket Zend Framework och en förklarning varför användning av ramverk är lönsamt. Zend Frameworks arkitektur och funktionalitet gås igenom för att få en förståelse för hur ramverket kan utnyttjas för att bygga webbapplikationer i praktiken.

I den praktiska delen undersöks funktionalitet som ingår i webbplatsen och i innehållshanteringssystemet bakom den. Tillämpning av den teoretiska delen lyfts fram och de flesta funktioner förklaras med hjälp av kodexempel för att understryka hur de har implementerats.

Den färdiga produkten blev en välfungerande webbapplikation med ett kraftfullt men enkelt innehållshanteringssystem vilket uppfyllde förväntningarna beställaren hade. Att använda ett ramverk för detta projekt var en väldigt bra val eftersom utvecklingen gick snabbare och blev enklare.

Nyckelord:	Zend Framework, Innehållshanteringssystem, Skräddarsydd Ramverk, Designmönster, MVC
Sidantal:	59
Språk:	Svenska
Datum för godkännande:	12.5.2011

DEGREE THESIS	
Arcada	
Degree Programme:	Information technology
Identification number:	3124
Author:	Kristoffer Parikka
Title:	Development of a Web Site and a Customized Content Management System for the Degree Program Film and Television at Arcada University of Applied Sciences
Supervisor (Arcada):	Hanne Karlsson
Commissioned by:	ARCADA FILM AND TV

Abstract:

This thesis is about developing a website with a customized content management system made for the degree program Film and Television at Arcada. The aim of the project was to create software needed by a website where Film and Television is able to present various disciplines and productions made by students.

The thesis consists of two parts. The theoretical section is a study of the design pattern MVC, framework and content management system; What they are used for and which main features they generally consist of. Moreover benefits that can be gained by using a tailor-made content management system are investigated. After this the Zend Framework is presented in detail and the advantage of using such a framework is explained. Zend Frameworks architectural design and features are reviewed to gain understanding of how the framework can be utilized to build practical web applications.

The practical section examines features that where developed for the website and the back-end content management system. Knowledge gained from the theoretical part is applied and mostly features are explained with code examples to emphasize how different functions have been implemented.

The delivered product is a well-functioning web application, with a powerful yet simple content management system which met the expectations the customer had. Using a framework for this project was a very good choice because the development was faster and easier.

Keywords:	Zend Framework, Content management system, Tailor made, Framework, Design pattern, MVC
Number of pages:	59
Language:	Swedish
Date of acceptance:	12.5.2011

INNEHÅLL

1 INLEDNING 8

1.1 Utgångspunkt 8
1.2 Målsättning 8
1.2.1 Den teoretiska delen 8
1.2.2 Praktiska delen 9
1.3 Begränsningar 9

2 DESIGNMÖNSTRET MVC 10

2.1 Definition 10
2.2 Historia 10
2.3 Designmönster inom datateknik 10
2.3.1 MVC 11

3 RAMVERK FÖR MJUKVARA 12

3.1 Definition 12
3.2 Varför används ramverk? 13
3.3 Olika webbramverk 13
3.3.1 PHP-ramverk 14
3.3.2 Andra webbramverk 14
3.3.3 Ramverk för klientdelen 14

4 ZEND FRAMEWORK 16

4.1 Allmänt 16
4.2 MVC 17
4.3 Autentisering och åtkomst 19
4.4 Internationalisering 20
4.5 Kommunikation mellan applikationer 21
4.6 Webbtjänster 21
4.7 Kärna 22
4.8 Installation och filstruktur 23
4.9 Zend Framework och Ajax 25
4.10 Hjälpklasser 25

5 INNEHÅLLSHANTERINGSSYSTEM 26

5.1 Definition 26
5.2 Funktionsprincip 26
5.2.1 Skapande av innehåll 27
5.2.2 Innehållshantering 27

5.2.3 Publicering av innehåll ... *27*
5.2.4 Presentationen av innehållet ... *28*
5.3 Krav ... 28
5.3.1 Säkerhet och integritet ... *28*
5.3.2 Autentisering ... *29*
5.3.3 Åtkomstkontroll ... *29*
5.3.4 Utvidgningsmöjlighet ... *29*
5.4 Skräddarsytt innehållshanteringssystem ... 29
6 HUR WEBBPLATSEN SKAPADES ... 31
6.1 Utvecklingsverktyg ... 31
6.2 Den publika webbplatsens uppbyggnad ... 33
6.2.1 Huvudkategorierna ... *33*
6.3 Flerspråkighet ... 34
6.4 Innehållet ... 36
6.4.1 Flash-banner ... *36*
6.4.2 Innehållet i textrutan ... *38*
6.4.3 Ansök nu ... *40*
6.4.4 Sidfot ... *41*
6.4.5 Undermenyer ... *42*
6.5 Innehållshanteringssystemet ... 43
6.5.1 Inloggning ... *43*
6.5.2 Innehållshanteringsmeny ... *44*
6.5.3 Menyhantering ... *45*
6.5.4 Redigera innehåll ... *48*
6.5.5 Redigera sidfoten ... *51*
6.5.6 Ansök nu ... *52*
7 DISKUSSION ... 53
7.1 Innehållshanteringssystem ... 53
7.2 Zend Framework ... 54
7.2.1 Fördelar ... *55*
7.2.2 Nackdelar ... *55*
7.3 Slutsatser ... 55
KÄLLOR ... 57
BILAGA 1: JÄMFÖRELSE av PHP-RAMVERK ... 59

Figurer

Figur 1. Funktionsprincipen i MVC (Programmer's Reference Guide: Zend Framework 2010). 11
Figur 2. MVC-arkitekturen i Zend Framework (Allen, Lo & Brown 2009:11). 17
Figur 3. Definitionsexempel på page-modellen i ”Film och Tv”-webbapplikationen. .. 18
Figur 4. Initialisering och användningsexempel för page-modellen i ”Film och Tv”-webbapplikationen. 19
Figur 5. Användning av authenticate-funktionen i ”Film och Tv”-webbapplikationen. 20
Figur 6. Exempel på syntaxen i en konfigurationsfil med ini-format. 22
Figur 7. Filen index.php samt demonstration av användning av konfigurationsfil. 23
Figur 8. WWW-applikationens filstruktur gjord med ZF-verktyget. 24
Figur 9 Ajax funktionsprincip 25
Figur 10 Zend Studio - Programmeringsgränssnittet 32
Figur 11 FlashDevelop - Programmeringsgränssnittet 33
Figur 12 Film & Tv - Framsidan 41
Figur 13 Film & Tv - Kategorisidan utbildningsprogrammet 43
Figur 14 Film & Tv – Inloggning 44
Figur 15 Innehållshanteringsmenyn 45
Figur 16 Menyhantering 46
Figur 17 Formulär för att skapa menyobjekt 48
Figur 18 Film & Tv – Innehållsredigering 50
Figur 19 Film & Tv – Sidfot-redigering 52
Figur 20 Film & Tv – ”Ansök nu”-knappens redigering 52

Förkortningar

API	Application Programming Interface
CMS	Content Management System
HTML	Hypertext Markup Language
JSON	JavaScript Object Notation
PHP	Hypertext Preprocessor
RSS	Really Simple Syndication
URL	Uniform Resource Locator
XML	eXtensible Markup Language
IDE	Integrated Development Environment
AJAX	Asynchronous JavaScript
SQL	Structured Query Language
LAMP	Linux, Apache, MySQL, PHP
GTW	Google Web Toolkit
CSS	Cascading Style Sheets
RBAC	Roll Based Access Control

1 INLEDNING

1.1 Utgångspunkt

Arbetet är gjort som ett projekt för utbildningsprogrammet Film och TV vid Arcada. Syfte med projektet var att skapa den program-tekniska delen av en webbplats som inte är integrerad med Arcadas webbportal, där utbildningsprogrammet i Film och Tv kunde presentera de olika inriktningsalternativ skolan har att erbjuda. Ett syfte var också att presentera produktioner gjorda av studerande vid utbildningsprogrammet i Film och Tv. Kraven var att webbplatsen inte skulle byggas upp med hjälp av något proprietärt innehållshanteringssystem eller någon öppen källkodsprodukt, såsom t.ex. Joomla eller Drupal. Det beställaren önskade var att webbplatsen skulle bli så enkel att upprätthålla som möjligt, men ändå innehållsmässigt flexibel och enkel att redigera. Projektet är uppbyggt med PHP-ramverket Zend Framework, JQuery, och webbplatsen upprätthålls i LAMP-miljö.

1.2 Målsättning

Målsättningen för examensarbetet var att utveckla en webbplats enligt en färdiggjord layout och att implementera ett skräddarsytt innehållshanteringssystem.
Den skriftliga delen av examensarbetet är indelad i en teoretisk och en praktisk del. Den teoretiska delen består av en beskrivning av hur Zend Framework är uppbyggt och hurudana funktioner och möjligheter som erbjuds. En analys om vad som krävs av ett innehållshanteringssystem ingår även. Den praktiska delen av examensarbetet beskriver utvecklingsprocessen; hur webbplatsen och det skräddarsydda innehållshanteringssystemet skapades.

1.2.1 Den teoretiska delen

Målsättningen med den teoretiska delen är att ge en djupare förståelse för hur ramverket Zend Framework är uppbyggt, hur det fungerar och vilka möjligheter som en

programmerare har. Eftersom examensarbetet handlar om att skapa en webbplats och ett skräddarsytt innehållshanteringssystem, är ett mål att även redogöra för:

- varför det är lönsamt att använda ett ramverk i projekt av detta slag
- vad som krävs av ett fungerande innehållshanteringssystem
- en jämförelse av vilka fördelar och nackdelar ett skräddarsytt innehållshanteringssystem har jämfört med proprietära eller öppen källkodssystem.

1.2.2 Praktiska delen

Målsättningen med den praktiska delen är att klarlägga hur man utvecklar en webbaserad applikation med hjälp av Zend Framework samt vilka andra tekniker som kan användas om något saknas i ramverket eller om man behöver åsidosätta funktioner man inte vill använda. Andra mål är att redogöra för de tekniska lösningarna bakom ”Film och Tv”-webbplatsen och hur innehållshanteringssystemet är uppbyggt.

1.3 Begränsningar

Examensarbetet kommer inte att omfatta den visuella utformningen av webbplatsen utan fokuserar på hur ramverk och andra programmeringsverktyg kan utnyttjas för att skapa kraftfulla tekniska lösningar.

2 DESIGNMÖNSTRET MVC

2.1 Definition

Inom arkitektur och programutveckling innebär designmönster ett generellt sätt att lösa problem som ofta förekommer, d.v.s. en teknik med vilken man kan katalogisera typiska problem och deras typiska lösningar. I programutveckling används designmönster speciellt inom objektorienterad design. Designmönster beskriver funktionsprinciper och kommunikationen mellan objekt och klasser som anpassas för varje enskilt fall. (Freeman 2004:1)

2.2 Historia

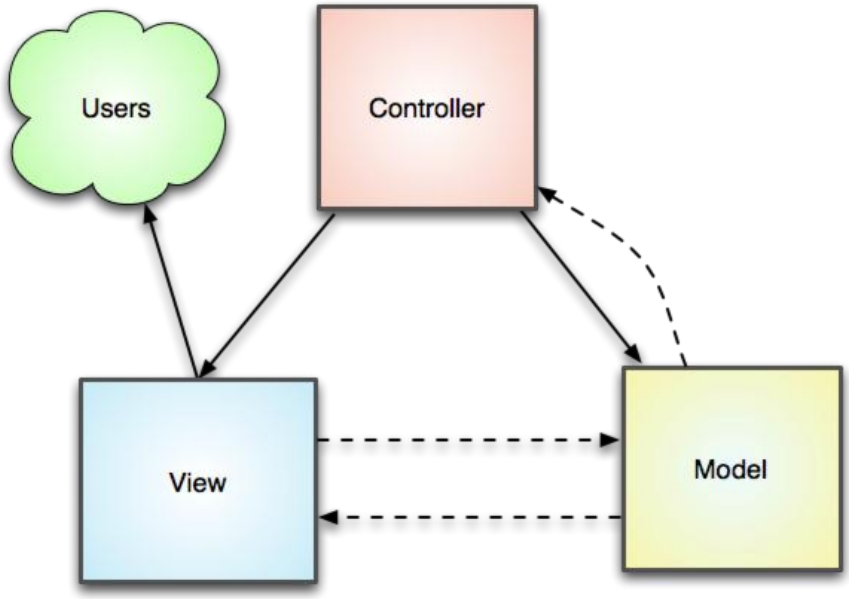
Idén med designmönster presenterades första gången av den amerikanska arkitekten Christopher Alexander i boken ”A Pattern Language”(1977) om formgivning av städer, byggnader och infrastrukturplanering.
År 1987 introducerades designmönster för första gången inom programutveckling av Kent Beck och Ward Cunningham vid OOPSLA-konferensen. Dessa teoretiska modeller var dock inte det samma som objektdesignmönstren i dagens bemärkelse. Snarare var det användningsmodeller för planering av objektorienterad programmering. Idéerna för dagens objektdesignmönster presenterades för första gången av Erich Gamma i sin doktorsavhandling år 1991. Senare väckte Gamma uppmärksamhet med den så kallade ”Gang of Four”-arbetsgruppens bok ”Design Patterns: Elements of Reusable Object-Oriented Software”. Den är fortfarande den mest kända boken om designmönster.

2.3 Designmönster inom datateknik

Designmönster används för behoven av en välplanerad programarkitektur inom datateknik för att sedan enkelt kunna underhålla och utvidga programvara.

2.3.1 MVC

MVC-designmönstret delar applikationen i tre olika delar: ”model, view, controller” eller fritt översätt till svenska: ”modell, vy, kontroller”. Denna indelning strävar efter separation av logik och vy. Med andra ord: användarens insats, avbildningen av den yttre världen och den visuella feedbacken till användaren separeras och hanteras som modell-, vy- och kontrollerobjekt. Kontrollern tolkar mus- och tangentbordsintryckningar från användaren, omvandlar dessa till kommandon som skickas till modellen och/eller vyn för att genomföra lämpliga förändringar i programmets tillstånd.



Figur 1. Funktionsprincipen i MVC (Programmer’s Reference Guide: Zend Framework 2010).

Modellen används för att hantera information och skicka påminnelser till observatörer när denna information ändras. Modellen innehåller endast data som är relaterat genom ett gemensamt mål. Till exempel om man har två grupper med ogemensam data och ogemensamma funktioner, skall två olika modeller skapas.

En modell sammanfattar mycket mer än bara data och funktioner som arbetar på den. En modell är avsedd att fungera som ett beräkningsprogram för approximation och abstraktion av verkliga processer och system. Den fångar inte bara tillståndet i en process eller system, utan också hur systemet fungerar. Detta gör det väldigt lätt att

använda verkliga beräkningsmodeller för att definiera egna modeller. Till exempel kan man definiera en modell för att överföra beräkningar från en administrativ vy till en vy som visas då applikationen används.

Vyn är ansvarig för att kartlägga grafik på en apparat. Vyn har oftast direkt motsvarighet till en skärm och vet därför hur visningen skall skötas. En vy fäster sig alltid till en modell och gör innehållet synligt på skärmen. Dessutom, när modellen ändras, omritas innehållet på skärmen automatiskt. Det kan finnas flera vyer för en och samma modell som kan visas på olika skärmar.

En vy kan också bestå av flera sub-vyer som kan själva innehålla flera sub-vyer.

Kontrollern är det medel vars hjälp användaren kommunicerar med applikationen. En kontroller accepterar inmatningar av användaren och styr sedan modellen och vyn att utföra åtgärder enligt begäran. I praktiken är det kontrollern som ansvarar för att omvandla slutanvändarens åtgärder till händelser i applikationen. Till exempel, om användaren klickar på musknappen eller väljer ett menyalternativ är kontrollern ansvarig för hur programmet ska reagera (Object orientation tips 1998).

3 RAMVERK FÖR MJUKVARA

3.1 Definition

Ett ramverk för mjukvara är programvara som innehåller en samling av funktioner och som tillsammans med den påbyggda koden bildar en färdig produkt. Vanligtvis kan inte ramverk användas som sådana, utan den färdiga produkten skapas genom att göra en ny applikation på ramverket. (Docforge 2010) Ramverken är ett specialfall av bibliotek för mjukvara genom att erbjuda återanvändbara abstraktioner av kod, insvept i en väl definierad API (Application Programming Interface). Ramverk innehåller några viktiga kännetecken som skiljer dem från vanliga bibliotek:

Inversion av kontroll: till skillnad från bibliotek eller normala användartillämpningar, dikteras inte programflödet av en metod eller ett objekt, utan av ramverket.

Standardbeteende: ett ramverk har ett standardiserat beteende. Dessa standardbeteenden måste vara användbara beteenden och inte bara en serie av operationer som saknar betydelse.

Tänjbarhet: ett ramverk kan utvidgas av användaren genom selektivt tvingande eller specialiserad användarkod som innehåller specifika funktioner.

Icke modifierbar ramverkskod: i allmänhet är det inte tillåtet att ändra ramverkets kod. Användaren kan utvidga koden men inte ändra. (Riehle 2000)

Det finns olika typer av ramverk inom mjukvaruindustrin: applikations, domän, plattform, komponent, service, utveckling etc. (Shan 2006)

3.2 Varför används ramverk?

Ramverk används för att göra det lättare och snabbare för en programmerare att skapa ny programvara. På grund av att ramverken innehåller färdiga delar för programvara som inte behöver kodas på nytt blir utvecklingstiden betydligt kortare. (Riehle 2000) Till exempel ett team som använder sig av ett webbapplikationsramverk för att utveckla en bankwebbtjänst, kan fokusera på hur ett kontouttag görs snarare än på mekanismen bakom hantering av begäran. (Shan 2006)

3.3 Olika webbramverk

Det så kallade webbramverket är redan en gammal företeelse. Ett av de första ramverken var skrivet för Gold Fusion och kallades ”Fusebox”. Ett annat var ”Struts” som var Java-baserat. Nyare betydelsefulla ramverk är ”Rails”. Rails-ramverket baserade sig på Ruby on Rails – språket som revolutionerade hela webbutvecklingen. (Wikipedia 2011)

3.3.1 PHP-ramverk

De första PHP-ramverken kom ut mellan år 2004-2005 och ett av dem var Zend Framework. Zend Framework (ZF) lanserades år 2005 på grund av den ökande populariteten av andra ramverk för programvara.(Webaxes 2010)
I dag finns det flera PHP ramverk som konkurrerar om titeln ”Det bästa PHP-ramverket”. Fastän Zend Framework är det mest använda webbramverket ännu idag är denna titel nästan omöjligt att ge, för åsikter om vilket är bäst finns nästan lika många som det finns PHP-ramverk. Jämförelse mellan olika PHP-ramverk hittas i bilaga 1.

3.3.2 Andra webbramverk

Det finns flera andra språk som används för webbprogrammering som också har ramverk. Det vanligaste är idag:

För Java

- **The Google Webb Toolkit** som kommer av morfar för alla Java-utvecklingsbutiker, Google. Detta ramverk är ett helt öppet källkodsprojekt som innehåller omfattande funktionalitet för Ajax-interaktion, och man kan enkelt skriva sin kod i Java som sedan kompileras av en GTW-kompilator (Google Web Toolkit) till bläddrarkompatibel JavaScrip och HTML
- **JSF ”**JavaServer Fases”-teknik förenklar skapandet av användargränssnitt för JavaServerapplikationer.

För C Sharp (C#)

- **.Net Framework** är Microsofts ramverk som byggs runt MVC-modellen. Programmeringsspråken för ramverket är antingen C#, C++ eller VisualBasic, men med utvidgningar kan man till exempel få tillgång till ramverkets funktioner med PHP.

3.3.3 Ramverk för klientdelen

Ramverk för klientdelen är specialiserade för gränssnitt/vy/presentation. Nyligen har ramverk för både JavaScript och CSS (Cascading Style Sheets) vunnit i popularitet.

Ett ramverk för klientdelen är oftast en mindre delmängd av ett större webbramverk. De populäraste idag är:

För JavaScript

- **JQuery**
- **Prototype**
- **Dojo**

För CSS

- **Yet Another Multicolumn Layout**
- **Blueprint**
- **Yahoo! UI Library: Grids CSS**

4 ZEND FRAMEWORK

4.1 Allmänt

Zend Framework är ett öppet källkods-projekt som siktar på att bli ett standardiserat ramverk som PHP-applikationer stöder sig på i framtiden. Ramverket kan beskrivas som ett PHP-bibliotek. Med ramverket är det enkelt att bygga fungerade, moderna applikationer som är enkla att underhålla tack vare att:

- Projektfilerna blir välorganiserade i mappar,
- Färdiga komponenter kan användas oberoende av andra komponenter i ramverket
- Ramverket fokuserar på kvalitetskod och uppdateras med jämna mellanrum
- Det är väldokumenterat.

(Techchorus 2009)

Zend Framework är uppbyggt så att ramverkets komponenter skall vara möjligast oberoende av varandra. Ramverket består av sex huvudkategorier:

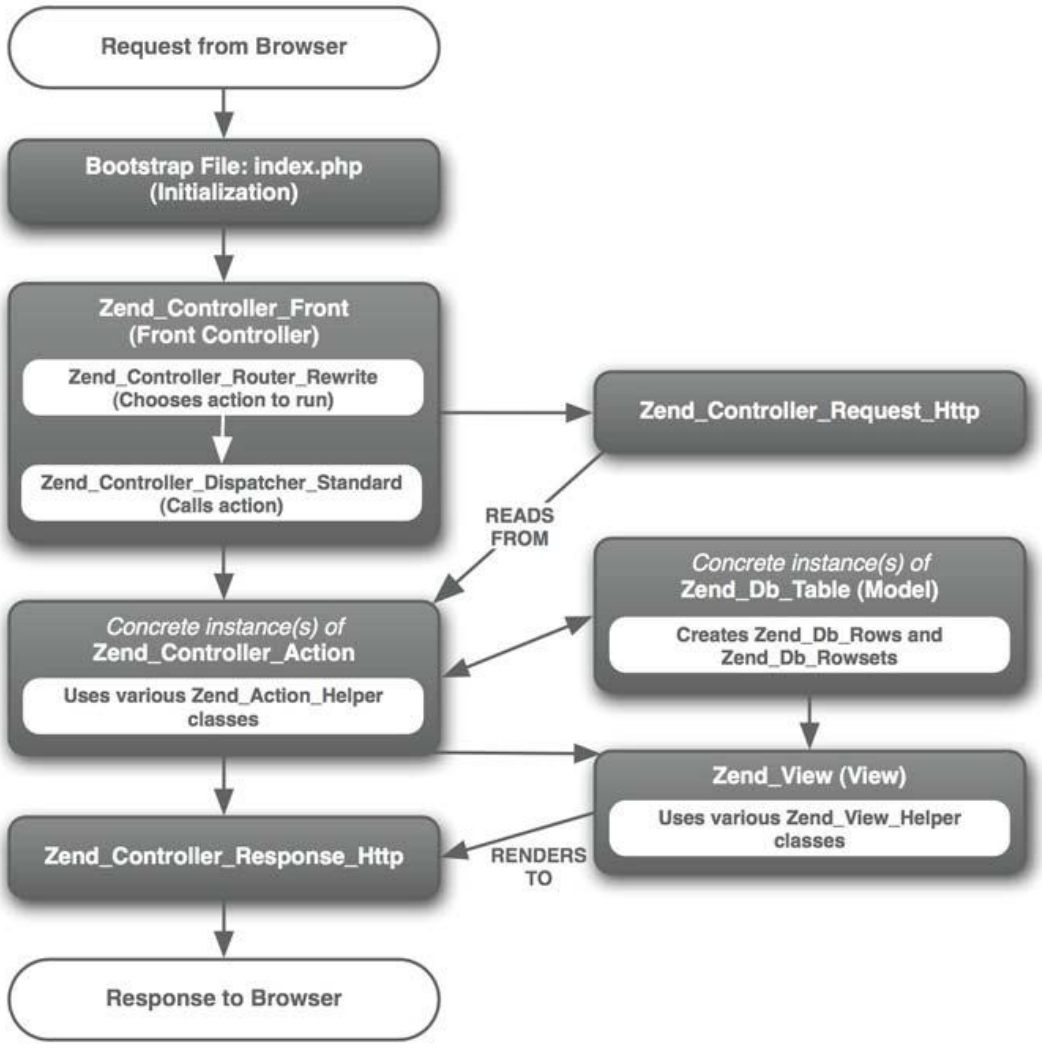
- MVC
- Autentisering och åtkomst
- Internationalisering
- Kommunikation mellan applikationer
- Webbtjänster
- Kärna.

(Allen, Lo & Brown 2009:10)

I de följande delkapitlena undersöks dessa närmare.

4.2 MVC

I Zend Framework byggs inte strukturen enligt MVC automatiskt utan den måste man skapa själv eller använda sig av programmet ZF gjort för detta ändamål (se kapitel 4.7). I Figur 2 demonstreras Zend Frameworks huvudfunktioner. Figuren visar hur Zend_Controller_Front får en sidoförfrågan och förmedlar den framåt. URL-hanteringen sker på ett helt annat vis i Zend Framework än i vanliga PHP-applikationer. Ett exempel på en vanlig URL kan vara: *index.php?controller=page &action=edit.* I Zend Framework finns en inbyggd förbehandlare som kan omvandla URL:en till ett format som är enkelt att komma ihåg: *http://filmochtv.arcada.fi/page/edit.*



Figur 2. MVC-arkitekturen i Zend Framework (Allen, Lo & Brown 2009:11).

Efter det sänds förfrågan från Zend_Controller_Front till Zend_Controllern_Action som kallas Zend Frameworks hjärta. Den är planerad så enkelt som möjligt för att vara enkel att modifiera. När en användare matar in en adress, kontrolerar Zend_Controllern, som är en instans av Zend_Controller_Action, om det finns sektorer efter domännamnet. Om textdelen efter domännamnet är till exempel *”page/edit”*, förstår Zend Framework att användaren vill redigera en sida. Varje textdel som kommer efter domännamnet innehåller en egen klass och metod. I detta exempel söker Zend_Controllern efter en klass, ”page”, och en metod, ”edit”. Om enbart klassnamnet anges, antar Zend Framework att index-funktionen skall användas. (Programmer’s Reference Guide: Zend Framework 2010a)

Zend_Db_Table som implementerar en databastabell, representerar modell-delen i MVC-arkitekturen. Modell-delen erbjuder affärslogiken för applikationen och är oftast databasbaserad. Med Zend_Db_Table får man en objektorienterad, databasoberoende tillgång till olika databaser såsom: Mysql, Postgres, SQL Server, Oracle och SQLite. (Allen et al. 2009:11)

Varje tabell i databasen har sin respektive modellklass. Figur 3 demonstrerar hur man skapar en modellklass för tabellen *”page”:*

```
1  <?php
2
3  class Model_Page extends Zend_Db_Table_Abstract {
4
5      protected $_name = 'pages';
6
7  }
8
9  ?>
```

Figur 3. Definitionsexempel på page-modellen i ”Film och Tv”-webbapplikationen.

Med hjälp av modellklassen kan man enkelt hämta data till kontrollerklassen som sedan behandlas och visas i bläddraren med HTML i en Zend_View. Figur 4 demonstrerar hur man utför ”SELECT * FROM page WHERE id = ’id’ ” i Zend Framework.

```
91          //initialisera ny modell
92          $pageModel = new Model_Page ();
93          //hämta data till modellen med id:n
94          $pageModel->find ( $id )
```

Figur 4. Initialisering och användningsexempel för page-modellen i ”Film och Tv”-webbapplikationen.

Dessa få rader hämtar en rad från databasen med en specifik id och sparar den i modellobjektet.

4.3 Autentisering och åtkomst

Det finns två inbyggda klasser i Zend Framework för användaridentifiering och verifikation (Zend_Acl och Zend_Auth). Zend_Acl är ett enkelt sätt att implementera en roll-baserad användarkontroll. Objektet som skall implementeras med användarkontroll registreras som en resurs genom att använda en färdig klass Zend_Acl_resource_Interface och funktionen getResourceID(). Zend_acl organiserar sedan resurserna till en hierarkisk träd-struktur till vilken man sedan kan skapa ärvda regler. Varje resurs kan tilldelas egna rättigheter (läs, skriv, uppdatering och redigering). En annan möjlighet är att konfigurera rättigheter så att de hör ihop med en specifik regel. Roller kan skapas genom att anropa Zend_Acl_Role_Interface och dess enda funktion getRoleId(). Roller kan ärva privilegier av andra roller.

Zend_Auth används för användaridentifikation. Användning av funktionaliteten kombineras oftast till Zend_Session. Implementeringen av klassen sker med Zend_Auth_Adapter_Interface som har en enda metod kallad ”authenticate”.

I Figur 5 demonstreras hur man anropar authenticate-funktionen. (Programmer’s Reference Guide: Zend Framework 2010b-c).

```
104             $data = $userForm->getValues ();
105             // sök default adapter
106             $db = Zend_Db_Table::getDefaultAdapter ();
107             //skapa auth adaptern
108             $authAdapter = new Zend_Auth_Adapter_DbTable ( $db, 'users', 'username', 'password' );
109             //sätt avnändarnamn och lösenord
110             $authAdapter->setIdentity ( $data ['username'] );
111             $authAdapter->setCredential ( md5 ( $data ['password'] ) );
112             //autentisera
113             $result = $authAdapter->authenticate ();
114             if ($result->isValid ()) {
115                 // spara data
116                 $auth = Zend_Auth::getInstance ();
117                 $storage = $auth->getStorage ();
118                 $storage->write ( $authAdapter->getResultRowObject ( array ('username',
119                                                                             'first_name',
120                                                                             'last_name',
121                                                                             'role' ) ) );
122                 return $this->_redirector->gotoUrl ( '/page/1' );
123             } else {
124                 $this->view->loginMessage = "Du gav fel användarnamn eller lösenord!";
125             }
```

Figur 5. Användning av authenticate-funktionen i ”Film och Tv”-webbapplikationen.

I Figur 5 på rad 108 skapar man ett objekt av auth-adaptern. Detta objekt används för att anropa authenticate-metoden som autentiserar användaren. Om autentiseringen lyckas skapas ett auth-objekt i vilket användarinformationen sparas. Ett felmeddelande visas för användaren om autentiseringen misslyckades.

4.4 Internationalisering

Zend Framework erbjuder också stöd för flerspråkighet. Komponenten Zend internationalisering erbjuder ett mycket omfattande bibliotek med olika metoder för att förverkliga flerspråkiga sidor, till exempel för visning av rätta symboler och valutor (Zend_Currency & Zend_Date) samt för översättning av innehållstexter till ett annat språk (Zend _Translate).

I det följande granskas funktionaliteten bakom Zend_Translate närmare.
Zend_Translate är Zend Frameworks sätt att skapa flerspråkiga applikationer. Enligt Zend Framework kan man dela in byggandet av en sajt i fyra olika skeden:

1. Välj vilken anpassare du vill använda
2. Skapa språkvyn och implementera Zend_Translate i din kod
3. Skapa ursprungskoden
4. Översätt ursprungsfil till valda språk

Det finns flera olika slags anpassare men den mest använda är Zend_Translate_Adapter_Gettext. (Programmer's Reference Guide: Zend Framework 2010d)

4.5 Kommunikation mellan applikationer

Klasserna som hör till kategori kommunikation mellan applikationer används för kommunikation mellan sajter. Klasserna fungerar mycket likt curl-utvidgningen i PHP dvs. som ett bibliotek med vilket man kan skapa förbindelser till olika servrar med hjälp av olika protokoll. Med hjälp av dessa klasser får man till exempel stöd för JSON protokollet genom att använda Zend_Json.

Zend_Json erbjuder ett lätt sätt att plocka dynamisk data från servrar innan det överförs till Ajax-tjänsten. Zend_Json erbjuder också funktionalitet för att översätta PHP till JavaScript och vice versa.

Zend_Json innehåller också annan inbyggd funktionalitet. Ett exempel är *"fromXml()"* som möjliggör översättning av XML till JSON. (Programmer's Reference Guide: Zend Framework 2010e)

4.6 Webbtjänster

Zend erbjuder en del funktioner som möjliggör användning av andra webbtjänster. Till exempel Zend_Feed kan användas för att handskas med RSS-flöden. Zend erbjuder också komponenter för att ansluta Twitter, Yahoo, Google och många andra publika API:n till en egen applikation. Till Googles tjänster har Zend Framework skapat en helt egen komponent, Zend_Gdata. Zend_Gdata är ett PHP5-gränssnitt. Med det kan man till exempel använda Google-tjänsterna blogg, kalender, youtube, kodsökning och Picasa. (Programmer's Reference Guide: Zend Framework 2010f)

4.7 Kärna

Kärnan innehåller en samling olika komponenter som inte kan grupperas i andra kategorier. I denna grupp finns till exempel klasser för att söka klassinformation (Zend_Search_Lucene), användning av mellanminne (Zend_Cacke), skapande av Pdf-filer (Zedn-Pdf) och e-post (Zend_Mail). En av de viktigaste klasserna i denna kategori är ändå Zend_Config.

Med Zend_Config kan man lätt och behändigt sköta konfigureringen för hela applikationen. Konfigurationerna kan vara i olika format, de vanligaste är Xml- och Ini-filer. (Programmer's Reference Guide: Zend Framework 2010g). Figur 6 visar ett exempel på en konfiguration med ini-format.

```
14 [db]
15 resources.db.adapter = "pdo_mysql"
16 resources.db.params.host = "localhost"
17 resources.db.params.username = "root"
18 resources.db.params.password = ""
19 resources.db.params.dbname = "filmochtv"
20 resources.db.isDefaultTableAdapter = true
```

Figur 6. Exempel på syntaxen i en konfigurationsfil med ini-format.

Ur Figur 6 framgår att:

- Rad 14 anger att nedanstående rader handlar om databaskonfigurationer.
- Rad 15 berättar att databasadaptern som skall användas är pdo_mysql.
- Rad 16-19 används för att kontakta databasen.
- Rad 20 anger om databasen skall användas automatiskt eller inte.

```php
<?php

// Define path to application directory
defined('APPLICATION_PATH')
    || define('APPLICATION_PATH', realpath(dirname(__FILE__) . '/../application'));

// Define application environment
defined('APPLICATION_ENV')
    || define('APPLICATION_ENV', (getenv('APPLICATION_ENV') ? getenv('APPLICATION_ENV') : 'production'));

// Ensure library/ is on include_path
set_include_path(implode(PATH_SEPARATOR, array(
    realpath(APPLICATION_PATH . '/../library'),
    get_include_path(),
)));

/** Zend_Application */
require_once 'Zend/Application.php';

// Create application, bootstrap, and run
$application = new Zend_Application(
    APPLICATION_ENV,
    APPLICATION_PATH . '/configs/application.ini'
);
$application->bootstrap()
            ->run();
```

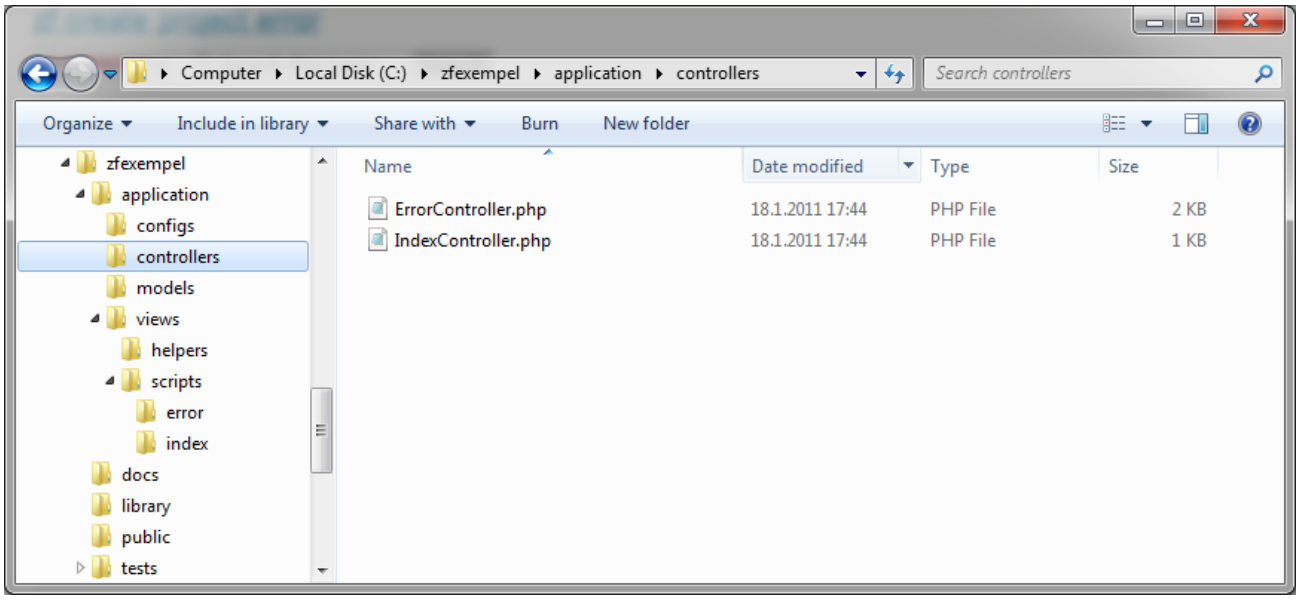
Figur 7. Filen index.php samt demonstration av användning av konfigurationsfil.

Figur 7 visar på rad 23 hur en konfigurationsfil definieras i Zend Framework. Detta görs i filen *”index.php”* som körs varje gång man skriver en förfrågan till någon del av webbplatsen. På rad 25 och 26 kan man se hur själva applikationen körs genom att anropa bootstrappern som definierar vilka resurser och komponenter som skall initialiseras. (Programmer’s Reference Guide: Zend Framework 2010g)

4.8 Installation och filstruktur

Installationen av Zend Framwork sker med programmet ZF men kan också göras manuellt. Med ZF-verktyget som används via kommandotolken är det ändå enkelt och snabbt. Med hjälp av det kan man minimera mänskliga fel till exempel när man skriver filnamn. Kommandoradsverktyget används inte bara för att skapa ett projekt, utan också för att lägga till moduler och klasser till applikationen i ett senare skede. Oavsett vilket sätt man väljer krävs katalogen ”library” i applikationens rot. Library-katalogen innehåller alla filer som är viktiga för att Zend Framework skall funktionera och därför är det viktigt att hålla katalogen utanför själva applikationen. (Allen, Lo & Brown 2009).

Applikationens olika delar uppdelas också i olika fysiska kataloger. Oftast uppdelas katalogstrukturen i fyra olika huvudkataloger men det kan variera. Vanligtvis räcker ändå strukturen som skapas med ZF-verktyget. (Pope 2009: 45)
Figur 8 visar filstrukturen samt kontrollers som är skapade med ZF-verktyget.



Figur 8. WWW-applikationens filstruktur gjord med ZF-verktyget.

Huvudkatalogen i zf-exempelapplikationen

Application-katalogen innehåller kataloger för kontrollers, vyer, modeller och konfigurationsfiler. I modulbaserade applikationer skall application-katalogen innehålla en modules-katalog som skall innehålla kataloger namngivna enligt modulens namn som sedan skall ha egna kontrollers, vyer, modeller och konfigurations-kataloger.

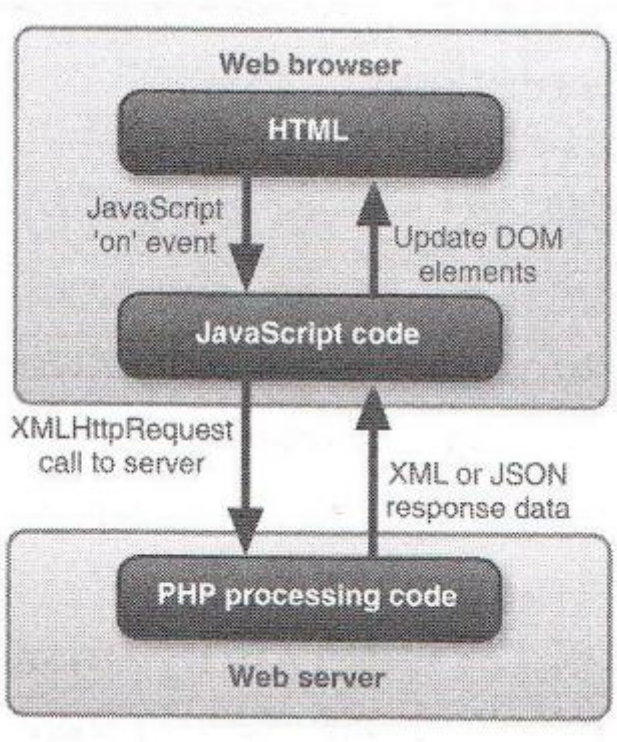
Public-katalogen innehåller applikationens *index.php*-fil, CSS-filer, media, JavaScript samt .htaccess-fil som används för att skriva om adresser.

Library-katalogen innehåller användarens egna komponenter som inte inkluderas i modellerna (Pope 2009, 13).

Test-katalogen innehåller applikations kod för testning.

4.9 Zend Framework och Ajax

En Ajax-förfrågan sker på samma sätt som andra funktionsförfrågningar och det behövs alltid en kontroller som tar emot den. Den enda skillnaden är att förfrågan returnerar JSON-snippet. Figur 9 förklarar funktionen närmare. För att behandla Ajax-förfrågningar kan man bygga hjälpfunktioner som behandlar informationen och uppdaterar vyn. (Allen, Lo & Brown 2009: 95-96.)



Figur 9 Ajax funktionsprincip

4.10 Hjälpklasser

Zend Framework innehåller många hjälpfunktioner. Dessa funktioner formar så kallade hjälpklasser vilka innehåller färdigt implementerade funktioner och koder som upprepas. Med hjälp av hjälpklasser sorteras generella och ofta använda funktioner på ett ställe, och då blir administrationen av applikationen mycket enklare. (Allen, Lo & Brown. 2009: 33 – 34.)

Med hjälpklasserna kan man till exempel enkelt hantera JSON-förfrågningar, Ajax-begäran och Flash-meddelanden. En av de viktigaste hjälpklasserna är ändå ViewRenderer vars huvudroller är:

- Skapa globala vy-objekt,
- Möjliggöra konstanta vyer för varje kontroller,
- Automatisera vy-objektens registrering till kontrollers.

(Programmer's Reference Guide: Zend Framework 2010h)

5 INNEHÅLLSHANTERINGSSYSTEM

I detta kapitel besvaras frågorna vad CMS (Content Managment System) är, hur det fungerar och vad som är nyttan med CMS i webbapplikationer. Innehållshanteringssystem är mer eller mindre nödvändigt i dagens läge på grund av stora och komplexa webbplatser med mängder av material vilka är svåra att underhålla.

5.1 Definition

Ett innehållshanteringssystem är en uppsättning av verktyg för att skapa, redigera och hantera innehåll på en webbplats. Med denna uppsättning kan man oftast hantera stora delar av webbplatsens funktionalitet, såsom t.ex. var innehållet skall visas, för vilka användare och hur många gånger innehållet visas, och vilket samband innehållet har till andra delar av webbplatsen. Det väsentliga med systemet är att det möjliggör hantering och administrering av innehåll för personer med sämre tekniska kunskaper.

5.2 Funktionsprincip

En av de viktigaste funktionerna i ett innehållshanteringssystem är att det skiljer åt webbplatsens design, innehåll samt affärslogiker och funktioner, vilket gör det enkelt att ändra på olika delar av webbplatsen utan att det påverkar på andra delar. (Peacock, 2009:12)

Innehållshanteringssystem använder sig av en viss typ av arbetsflöde. Detta kan variera mellan olika system men den grundläggande funktionella principen kan dock delas in i fyra kategorier:

- skapande av innehåll
- innehållshantering
- publicering av innehåll
- presentation av innehåll

5.2.1 Skapande av innehåll

Innehållshanteringssystem som är gjorda för webben brukar ha verktyg såsom texteditorer (som liknar vardagliga texteditorer) som hjälper användarna att skapa och redigera innehåll för webbplatsen utan att kräva några webbprogrammeringskunskaper. Detta gör att innehållshanteringssystem är så efterfrågade och allmänna i dag.

5.2.2 Innehållshantering

Allt innehåll och all annan relevant information som till exempel stil, kategorier, taggar eller användaridentifikation sparas i en databas av något slag. Detta leder till att material inte är kopplat till någon typ av presentationslager. Materialet brukar sparas som råtext i databasen. Därför är det möjligt att presentera materialet dynamiskt och att kunna ändra på andra delar i systemet utan att påverka innehållet. Detta betyder att det är möjligt att omorganisera webbplatsen vid behov och göra kopplingar mellan användare i systemet för att kunna dela material.

5.2.3 Publicering av innehåll

Innehållshanteringssystem brukar ha mycket avancerade funktioner för publicering. Dessa funktioner kan oftast automatisk applicera önskat utseende och layout. Eftersom systemet ser till att innehållet ser likadant och konsekvent ut, kan användarna

koncentrera sig på att skriva innehållet utan att behöva sörja för utseendet. I innehållshanteringssystem har man också ofta funktioner som möjliggör att användaren kan spara material utan att publicera det för allmänheten på webbplatsen. Detta kan vara bra om man inte hinner skriva färdigt det man höll på med och vill fortsätta senare.

5.2.4 Presentationen av innehållet

I ett innehållshanteringssystem presenteras innehållet mycket dynamiskt. Detta betyder att webbsidan byggs upp med hjälp av på förhand definierade mallar. Mallen kan bestämmas av slutanvändaren eller av webbplatsens administratör som har bestämt hur materialet skall formateras innan det presenteras. Den slutliga webbsidan består av innehållet från databasen och med hjälp av designmallen byggs sidan upp.

Beroende på arkitekturen innehållshanteringssystemet använder, kan sidan byggas upp på olika sätt. Ett sätt är att allt material samlas och presenteras därefter. I vissa system finns en så kallad cache-funktion som används för att lagra ofta laddade sidor i en databas för att de snabbare ska kunna visas upp.

5.3 Krav

Ett innehållshanteringssystem går inte att tillämpa i alla situation man stöter på. Genom att undersöka arkitekturen i systemet kan man få veta var det är användbart och för vad det är ämnat. Oavsett hur arkitekturen ser ut, måste innehållshanteringssystemet dock svara mot några grundläggande krav.

5.3.1 Säkerhet och integritet

I dag finns det många hot mot webbplatser. Trots att grundläggande webbprotokoll som används idag begränsar detta, finns det alltid till exempel en risk med webbsidor där information ges genom användarinteraktion. Denna information måste skyddas från kapning eller modifiering. För att få det bästa skyddet måste säkerheten vara inbyggd i systemet från första början. Det skall också vara möjligt att enkelt säkra utvidgningar och därför är en standardiserad felhanteringsmekanism nödvändig.

5.3.2 Autentisering

Innehållshanteringssystem måste innehålla någon slags kontroll av användare via t.ex. autentisering. Det måste finnas stöd för att hantera allt från enstaka administrativa användare till att hantera tusentals olika användare och olika system för autentisering. Därför måste systemet vara så flexibelt som möjligt för att minska koden som behövs för detta. (Brampton 2008:10)

5.3.3 Åtkomstkontroll

Åtkomstkontroll behövs för att begränsa vem som till exempel kan konfigurera webbplatsen. Därför krävs det många olika rättigheter som tilldelas till olika grupper. Det mest populära sättet för detta är något som kallas rollbaserad åtkomstkontroll (RBAC – Roll Based Access Control). Detta innebär att rollerna ges behörigheter och användare tilldelas roller. (Brampton 2008:11)

5.3.4 Utvidgningsmöjlighet

För att kunna använda ett innehållshanteringssystem i flera system måste det vara enkelt att utvidga. Det borde inte finnas någon funktion som krävs i varje webbplats och i bästa fall tas nästan allting bort. Varje synlig funktion som läggs till kan ses som en utvidgning. Det visar sig att det ofta finns olika typer av utvidgningsmöjligheter när kraven för att bygga en webbplats anges. (Brampton 2008:11)
Detta är dock inte så viktigt i innehållshanteringssystem som är skräddarsydda för en enda webbplats.

5.4 Skräddarsytt innehållshanteringssystem

I dag använder de flesta större webbplatser någon typ av innehållshanteringssystem. Flera webbplatser väljer ändå att använda skräddarsydda innehållshanteringssystem vilket betyder att de är enkom gjorda för ifrågavarande webbplats. Orsaker till att välja ett skräddarsytt innehållshanteringssystem kan vara till exempel:

Unik funktionalitet: I fall webbplatsen är speciell och unik jämfört med allmänna sajter kan det vara enklare att skapa ett skräddarsytt system än att börja ändra ett existerande.

Enkla webbplatser: Proprietära och innehållshanteringssystem med öppen källkod brukar innehålla mycket omfattande funktioner som kan kännas överväldigande för många. Ett skräddarsytt system kan göras mycket enkelt genom implementering av endast nödvändig funktionalitet.

Prestanda: Ett skräddarsytt innehållshanteringssystem som är optimerat för en enda webbplats är nödvändigt när man behöver hög prestanda. Proprietära och mjukvarulösningar med öppen källkod kan orsaka förminskad prestanda på grund av onödig funktionalitet och icke optimerad kod. Prestanda behövs när man har en hög belastning (dvs. många användare samtidigt) på webbplatsen.

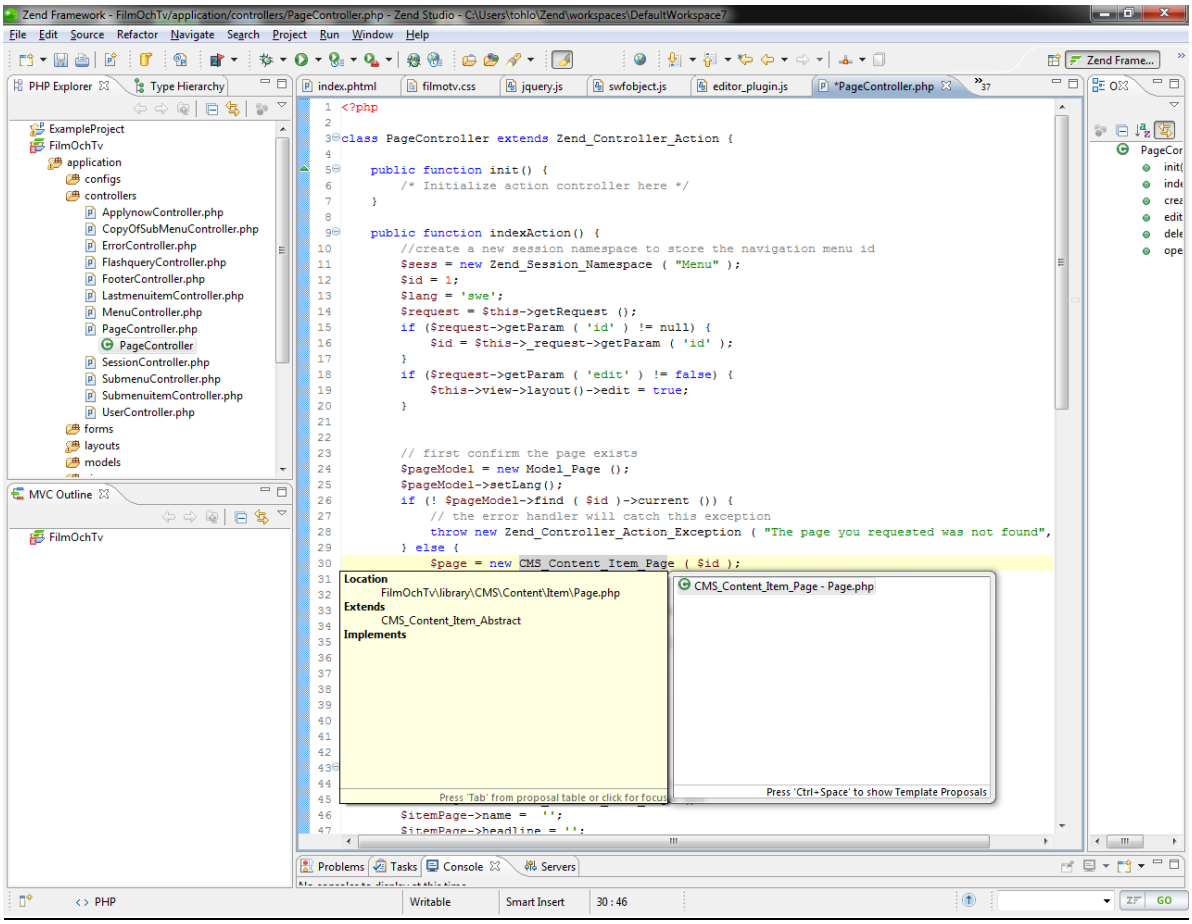
6 HUR WEBBPLATSEN SKAPADES

Den praktiska delen av examensarbete gick ut på att utveckla funktioner för en webbplats med en färdig layout och ett skräddarsytt innehållshanteringssystem. Detta kapitel redogör för vilka verktyg som användes, hur webbplatsen samt innehållshanteringssystemet är uppbyggt och de mest centrala funktioner som skapades. Exempelkoden är i stort sett såsom den är implementerad i projektet men vissa funktioner är förenklade för att ge en bättre helhetsbild.

6.1 Utvecklingsverktyg

För programmeringen av webbplatsens PHP, HTML, JavaScript och CSS användes Zend Studio version 7.2.1 som är byggd på öppen källkodsprojektet Eclipse. Zend Studio är en PHP IDE (Integrated Development Environment) som är ämnad för programmering med Zend Framework.
Zend Studion valdes som programmeringsverktyg för att den erbjuder bl.a. syntaxfärgläggning för PHP, CSS, HTML och JavaScript. Den har också andra smarta egenskaper som underlättar programmeringen såsom lokal- och fjärrdebuggning för PHP och JavaScript, automatisk kodkomplettering och enkel funktionsnavigering.

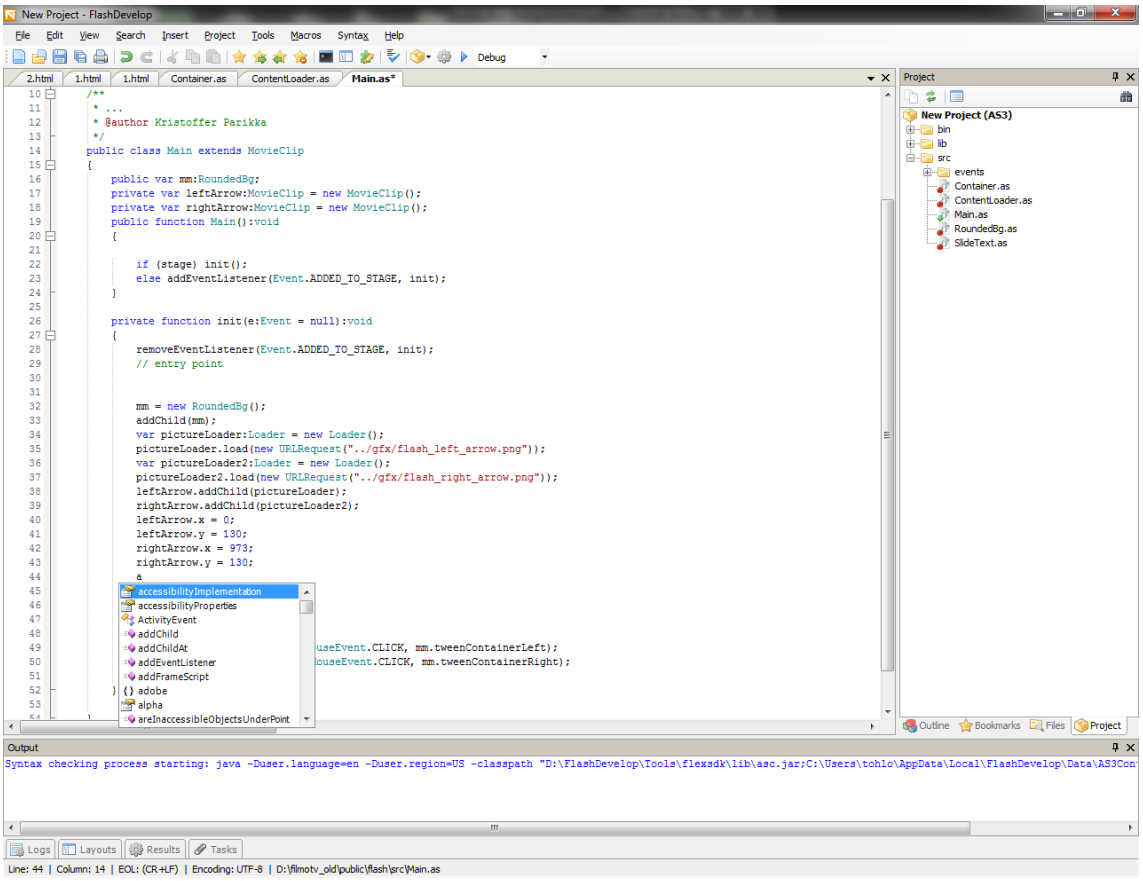
I Figur 10 visas programmeringsgränssnittet i Zend Studio med syntaxfärgläggning, automatisk kod-komplettering och funktionsnavigering. Koden i figuren är en del av PageController-klassen för webbplatsen.



Figur 10 Zend Studio - Programmeringsgränssnittet

För programmering av framsidans flash-banner (se kapitel 6.5) användes FlashDevelop. FlashDevelop är ett programmeringsverktyg med öppen källkod för ActionScript med liknande egenskaper som Zend Studio har för PHP.

I Figur 11 visas programmeringsgränssnittet i FlashDevelop med syntaxfärgläggning, automatisk kod-komplettering och funktionsnavigering. Koden är en del av Main-klassen i flash-bannern.



Figur 11 FlashDevelop - Programmeringsgränssnittet

FlashDevelop valdes för att den är gratis och för att den innehöll hjälpverktyg som inte fanns i Adobe Flash, såsom till exempel kod-komplettering.

6.2 Den publika webbplatsens uppbyggnad

6.2.1 Huvudkategorierna

Webbplatsen består av sju huvudkategorier:

- Framsida
- Utbildningsprogrammet
- Studiemiljö
- Personal
- Studentproduktioner
- Att studera på Arcada

- Samarbete

Varje huvudkategorilänk skapas med hjälp av klassen ”MenuController” genom att anropa den i ’layout.phtml’-filen. Layout-filen är en fil som alltid anropas när någon anropar en kontroller-klass. Stil och funktionalitet knyts ihop i denna fil. Själva anropet av kontrollern sker med hjälp av ”Zend_View_Helper_Action” som returnerar färdigt renderat innehåll av en kontroller-action.

echo $this->action ('index', 'menu', null, array ('menu' => $this->layout ()->page));

Denna rad anropar indexAction-metoden från ”MenuController” klassen med en parameter som berättar för klassen vilken huvudkategori skall visas som aktiv. En aktiv huvudkategori visas med en vit bakgrund i navigeringsbalken. Själva kontroller-klassen hämtar statiska data från databasen med de olika huvudkategorinamnen som sparas som en lista i kontrollern. Varje objekt i listan byggs sedan till en länk enligt den angivna layouten i en separat vy fil, som varje Action-metod har färdigt genererad av ZF-programmet (se kapitel 4.8). Vy-filen för indexAction-metoden heter alltså ”index.phtml.

6.3 Flerspråkighet

Varje huvudkategori är tillgänglig på svenska, finska och engelska. Språket går lätt att byta via respektive länkar på webbplatsen (se Figur 12). Varenda länk för språkval anropar indexAction-metoden i kontroller-klassen ”SessionController” med en parameter som berättar vilket språk som användaren valt.

```
public function indexAction() {

            $sess = new Zend_Session_Namespace ( "Lang" );
            //request language
            $lang = $this->_request->getParam ( 'lang' );
            //request pageid
            if($this->_request->getParam ( 'id' ) != ''){
                  $pageId = $this->_request->getParam ( 'id' );
            }
            else{
                  $pageId = 1;
            }
            //switch between languages
            switch ($lang) {
```

```
                case "se" :
                      $sess->lang = $lang;
                      break;
                case "fi" :
                      $sess->lang = $lang;
                      break;
                case "en" :
                      $sess->lang = $lang;
                      break;

          }
      return $this->_redirector->gotoUrl ( '/page/1' );
      }
```

Metoden skapar en ny instans av Zend_Session_Namespace-klassen som returnerar ett objekt som är bundet till en särskilt isolerad sektion av HTTP-sessionen. Denna sektion är namngiven med namnet ”Lang” som ges som en parameter till klassen. Efter att sessionen är sparad används hjälp-metoden ”Redirector” (se kapitel 4.10) för att ta användaren till framsidan som visas enligt det valda språket. Språket byts av SetLang-metoden som alltid körs först när en modell-klass anropas från kontrollern. Metoden avgör vilket språk skall användas genom att läsa sessions-objektet ”Lang”.

```
public function setLang() {
            //checks the language session for language
            $sess = new Zend_Session_Namespace ( "Lang" );
            //switch between language database
            switch ($sess->lang) {
                  case "fi" :
                        $this->_name = 'menus_fi';
                        break;
                  case "en" :
                        $this->_name = 'menus_en';
                        break;
                  default :
                        $_name = 'menus_se';

            }
      }
```

Ett annat alternativ för flerspråkiga sidor kunde ha varit Zend_Translate (se kapitel 4.4), men på grund av att materialet inte skulle bli helt identiskt för de olika språken, valdes alternativet att göra en egen lösning.

6.4 Innehållet

6.4.1 Flash-banner

Varje kategorisida är uppbyggd på samma sätt utom webbplatsens framsida. På framsidan ingår av en Flash-banner, med roterande presentationer av olika linjer på Film & Tv.

Flash-bannern är programmerad med ActionScript som är ett objekt-orienterat språk ursprungligen utvecklad av Macropmeda Inc. och idag ägs av Adobe Systems (Adobe 2005).

Flash-bannern består av följande ActionScript-klasser: Main, Container, ContentLoader, RoundedBg och en PHP-klass FlashqueryController.

För att kunna starta Flash-bannern, behövs klassen Main. Main-klassens uppgift är att skapa ett nytt objekt av RoundedBg-klassen och att skapa navigeringspilarna.

RoundedBg-klassen sköter om genereringen av Container-objekt och logiken för hur navigation animeras. För att kunna använda RoundedBg som ett synligt objekt måste den härleda MovieClip-klassen för att kunna använda de olika färdiga funktionerna som ActionScript bjuder på för enkel användning av synliga objekt.

För att hämta innehåll till Flash-bannern, används klassen ContentLoader, vilken härleder ”ActionScript Loader”-klassen som innehåller funktioner för att ladda information till Flash. Hämtningen av data sker med ActionScript-klasserna URLRequest() och URLLoader().

```
var request:URLRequest =
new URLRequest("http://filmochtv.arcada.fi/flashquery/index");
var variables:URLLoader = new URLLoader();

variables.dataFormat = URLLoaderDataFormat.VARIABLES;

variables.addEventListener(Event.COMPLETE, completeHandler);
try
{
```

```
      variables.load(request);

}
catch (error:Error)
{
      trace("Unable to load URL: " + error);
}
```

För att kunna ändra innehållet utan att gå in i ActionScript koden, laddas innehållstexter och bilder från externa filer. Innehållstexten i HTML-format (Hypertext Markup Language) och bilderna för innehållet hämtas till klassen genom anrop av en URL (Uniform Resource Locator) till PHP-klassen FlashqueryControllers indexAction()-metod.

```
public function indexAction()
    {

      $sess = new Zend_Session_Namespace ( "Lang" );
            //request language

            //switch between languages
            switch ($sess->lang) {
                  case "se" :
                        $dir = '../public_html/flash/bin/content/';
                        $flashdir =
'http://filmochtv.arcada.fi/flash/bin/content/';
                        break;
                  case "fi" :
                        $dir = '../public_htm/flash/bin/content_fi/';
                        $flashdir =
'http://filmochtv.arcada.fi/flash/bin/content_fi/';
                        break;
                  case "en" :
                        $dir = '../public_htm/flash/bin/content_en/';
                        $flashdir =
'http://filmochtv.arcada.fi/flash/bin/content_en/';
                        break;
                   default:
                        $dir = '../public_htm/flash/bin/content/';
                        $flashdir =
'http://filmochtv.arcada.fi/flash/bin/content/';
            }
            //directory to pictures

            $filecount = count(glob("" . $dir . "*.jpg"));

            // open specified directory

            $returnstr['directory'] = $flashdir;

            for( $count = 1; $count<= 5; $count++ ){
                  $returnstr['picture'.$count] = urlen-
code($count.".jpg");
```

```
                $returnstr['texts'.$count] = urlen-
code($count.".html");
            }

            header ("Content-Type: application/x-www-urlformencoded");
            //disable zend layout
            $this->_helper->layout->disableLayout();
            //send query to view
            $this->view->querystring = http_build_query($returnstr);
    }
```

FlashqueryController sköter om att skicka tillbaka HTML-kodade variabler av innehållstexter som är inlästa från HTML-filer samt bildadresser för bilder för att kunna ladda in bilderna för innehållet i ett senare skede. För att kunna anropa en Zend kontroller som bara skickar specifikt data, måste man åsidosätta laddningen av layouten. Detta sker genom att anropa Layout-hjälparklassens disableLayout-metod.

```
$this->_helper->layout->disableLayout();
```

Den hämtade innehållet måste omformas till olika Flash-objekt före de kan visas i flash-bannern. Detta sker i Container-klassen, som laddar in den begärda bilden samt ändrar innehållstexten till ett Flash-textobjekt. Container-klassen fungerar samtidigt som en behållare för innehållet som sedan kan styras från RoundedBg-klassen utan att innehållets formatering går sönder.

6.4.2 Innehållet i textrutan

Framsidan består också av en textruta där till exempel nyheter kan visas. Textrutan förekommer också på de övriga sidorna men i ett större format. Textrutan skapas genom att från Layout-filen anropa applikationens viktigaste klass ”PageController” och metoden indexAction.

```
public function indexAction() {
      //create a new session namespace to store the navigation menu id
      $sess = new Zend_Session_Namespace ( "Menu" );
      $id = 1;
      $lang = 'swe';
      $request = $this->getRequest ();
      if ($request->getParam ( 'id' ) != null) {
            $id = $this->_request->getParam ( 'id' );
```

```
	}
	if ($request->getParam ( 'edit' ) != false) {
		$this->view->layout()->edit = true;
	}
	// first confirm the page exists
	$pageModel = new Model_Page ();
	$pageModel->setLang();
	if (! $pageModel->find ( $id )->current ()) {
		// the error handler will catch this exception
		throw new Zend_Controller_Action_Exception ( "The page you
requested was not found", 404 );
	}
	else
	{
		$page = new CMS_Content_Item_Page ( $id );
		$array[] = $page->toArray();

		$this->view->layout()->page = $array[0]['menuid'];
		//store the id from page menuid
		$sess->menu = $array[0]['menuid'];
		$this->view->page = $page;
		$this->view->layout()->id = $id;
	}
```

När metoden anropas, anges framsidans id, om inte en annan sida har blivit efterfrågad. Efter att man vet vilken id sidan som skall visas till användaren har och kontrollerat att sidan finns i databasen, skapas en objekt av ”CMS_Content_Item_page”-klassen som ärver klassen CMS_Content_Item_Abstract, som har alla funktioner för att bygga upp innehållet. Id:n skickas till konstruktorn för ”CMS_Content_Item_Abstract”-klassen som en parameter som används för att söka sidans referens data från ”Page”-databastabellen med hjälp av ”loadPageObject()”-metoden.

```
public function __construct($pageId = null) {
		$this->_pageModel = new Model_Page ();
		$this->_pageModel->setLang();
		if (null != $pageId) {
			$this->loadPageObject ( intval ( $pageId ) );
		}
	}

public function loadPageObject($id) {
		$this->id = $id;
		$row = $this->_getInnerRow ();
		if ($row) {
			if ($row->namespace != $this->_namespace) {
				throw new Zend_Exception ( 'Unable to cast
page type:' . $row->namespace . ' to type:' . $this->_namespace );
			}
			$this->name = $row->name;
			$this->parent_id = $row->parent_id;

			$contentNode = new Model_ContentNode ();
			$contentNode->setLang();
```

```
                  $nodes = $row->findDependentRowset ( $contentNode );
                  if ($nodes) {
                        $properties = $this->_getProperties ();
                        foreach ( $nodes as $node ) {
                              $key = $node ['node'];
                              if (in_array ( $key, $properties )) {
                                    // try to call the setter method
                                    $value = $this->_callSetterMethod
( $key, $nodes );
                                    if ($value === self::NO_SETTER) {
                                          $value = $node ['content'];
                                    }
                                    $this->$key = $value;
                              }
                        }
                  }
            } else {
                  throw new Zend_Exception ( "Unable to load content
item" );
            }
      }
```

Metoden ”loadPageObject()” söker sedan sidans innehåll från databastabellen ”content_nodes” med data från Page-tabellen och skapar ett objekt av hela innehållet. Objektet innehåller data för att visas rätta under-menyer (se kapitel 6.4.5) samt innehållet för webbsidan. Själva innehållet visas redan i ”index Page”-vyn. ”Index Page”-vyn är delad i två delar med en if() sats, som avgör om sidan är framsidan eller någon av de andra sidorna som skall visas. Denna del av koden representerar presentationslagret i ett innehållshanteringssystem (se kapitel 5.2.4).

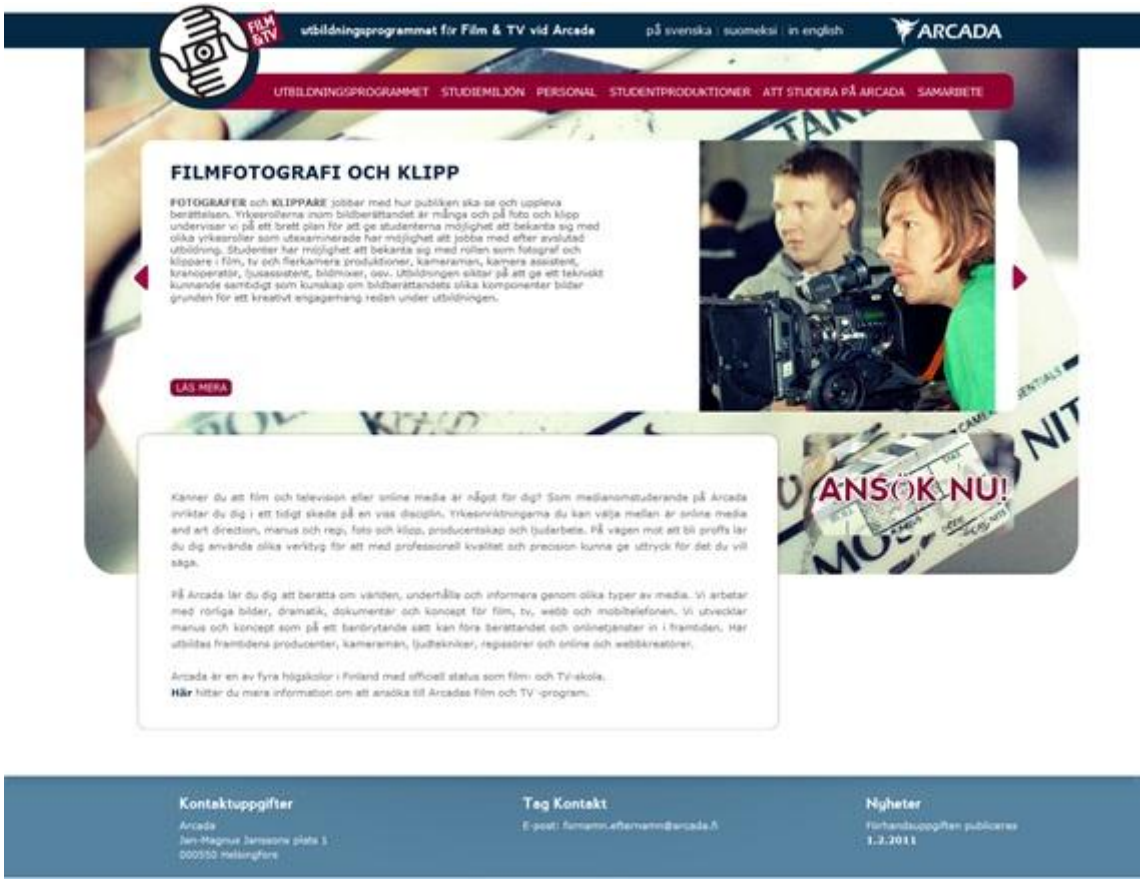
6.4.3 Ansök nu

En ”ansök nu”-knapp förekommer också på framsidan. Knappens synlighet och länk kan konfigureras i innehållshanteringssystemet (se kapitel 5.6.5). Knappen visas genom att anropa indexAction-metoden från ApplynowController-klassen med hjälp av Zend_View_Helper-klassen. Skillnaden med visningen av knappen är att den anropas redan från ”index Page”-vyn.

6.4.4 Sidfot

Sidfoten består av en balk med kontaktuppgifter och nyheter. Denna balk existerar på samma sätt på webbplatsens varje sida. Sidfoten anropas från Layout-filen på samma sätt som till exempel innehållet (se kapitel 6.2.1). FooterController-klassens indexAction-metod söker data från footer-databastabellen med hjälp av Application_Model_DbTable_Footer-klassen. När datat är samlat skickas det till footerns index-vy där utseendet genereras.

I Figur 12 visas ovannämnda komponenter och hur de är placerade på framsidan.



Figur 12 Film & Tv - Framsidan

6.4.5 Undermenyer

På varje sida, med undantag av framsidan finns en meny bredvid textrutan (se kapitel 6.4.2). Menyn fungerar som navigation för underkategorierna i varje huvudkategori. Varje meny hör till en förälderkategori för att systemet ska veta vilken meny som skall visas.

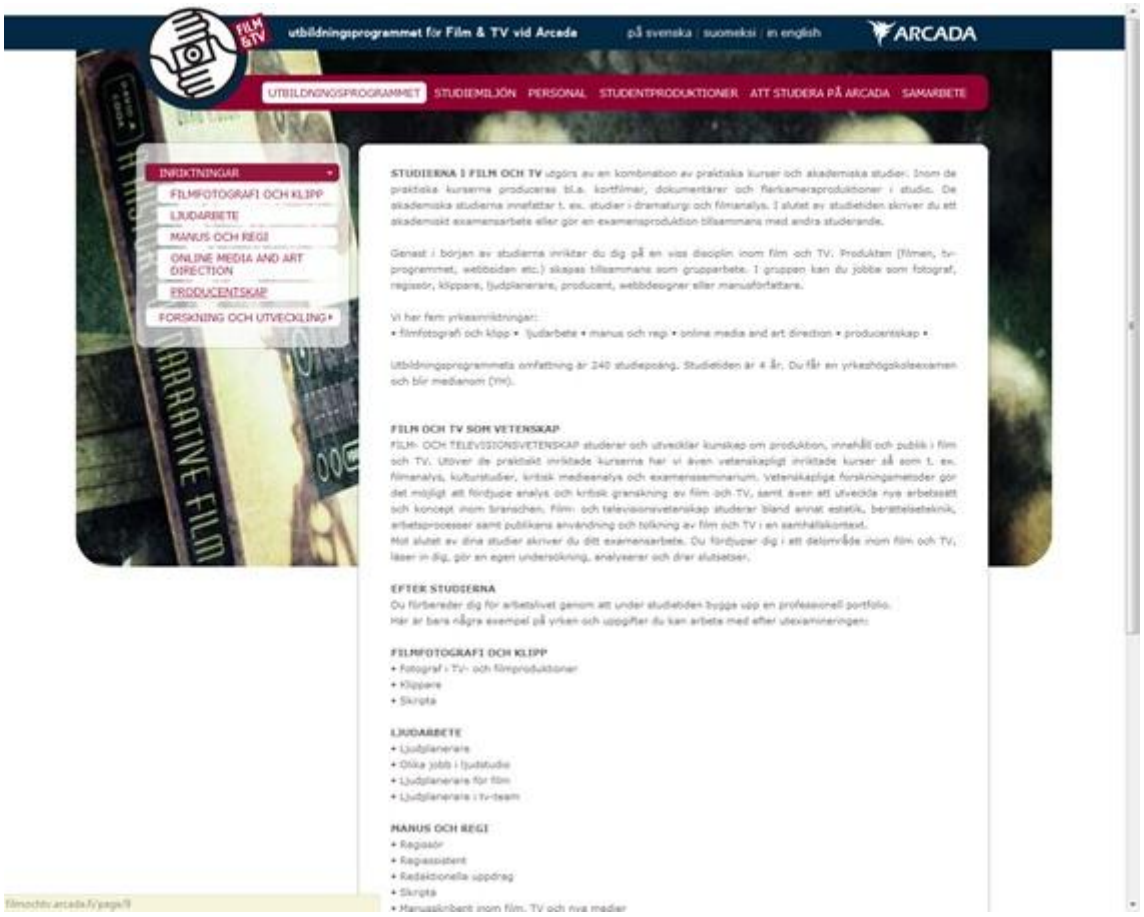
Menyn fungerar som ett ”träd” med tre nivåer bestående av menyobjekt. Varje meny-nivå har egna kontroller- och modell-klasser. Klasserna för varje nivå är nästan identiska förutom översta nivåns kontrollerklass som har en renderAction()-metod, vilken anropas från ”index Page”-vyn då en sida blir visad. Metoden ser till att rätt meny skapas.

```
public function renderAction() {
      $menu = $this->_request->getParam ( 'menu' );
      $pageId = $this->_request->getParam ( 'pageId' );

      //create Menu
      $navigation = new Application_Model_MenuNavigation();
      //return it to view
      $this->view->nav = $navigation->createMenu($menu,$pageId);

}
```

Själva menyn skapas i MenuNavigation-klassen som söker rekursivt alla barn-objekt för alla nivåer samt skapar dynamiskt HTML-koden för menyobjekten som sedan returneras till översta nivåns vy där menyn visas.

Varje menyobjekt i menyn kan fungera antingen som en länk som öppnar den undre nivån eller som en vanlig länk som laddar eftersökt innehåll för användaren. Animationen för att öppna och stänga olika nivåer är implementerad med ”JQuery treeview”-instickskprogramkod.

I Figur 13 visas en av huvudkategorisidorna med en meny som innehåller två nivåer av menyobjekt.

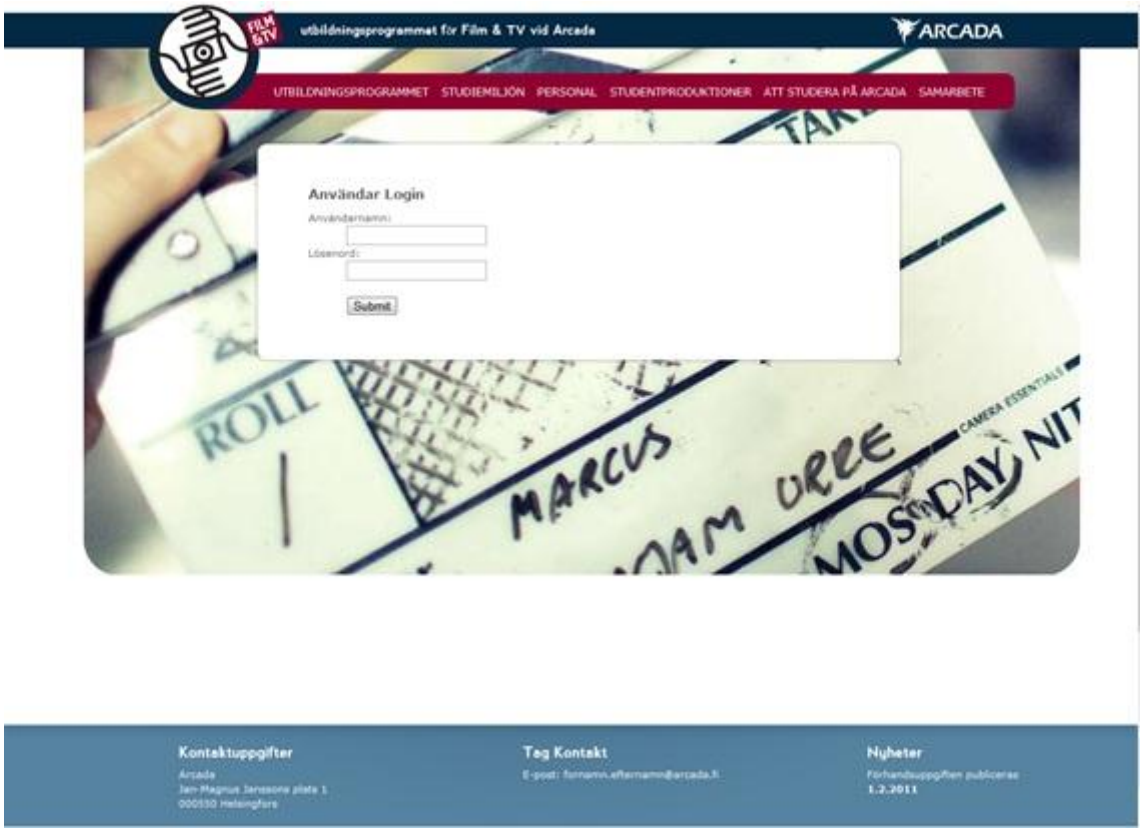


Figur 13 Film & Tv - Kategorisidan utbildningsprogrammet

6.5 Innehållshanteringssystemet

6.5.1 Inloggning

Inloggningen till innehållshanteringssystemet är helt och hållet gömd för normala användare. Det enda sättet att komma in är genom att ange en adress. Inga länkar till denna adress finns på den publika sidan. Inloggningen till innehållshanteringssystemet visas i Figur 14.



Figur 14 Film & Tv – Inloggning

Inloggningen sköts med hjälp av UserController-klassens loginAction()-metod (se Figur 5), som anropar User-formuläret. Själva valideringen av användaren och åtkomstkontroll sköt med Zend_Auth och Zend_Acl (se kapitel 4.3).

6.5.2 Innehållshanteringsmeny

Efter inloggningen kommer man till en likadan vy som på den publika sidan. Enda skillnaden är att innehållshanteringsmenyn (visas i Figur 15) dyker upp när man för musen till skärmens övre kant. Implementeringen av menyns animationer är skapade med funktionerna JQuery fadeIn() och fadeOut().

Figur 15 Innehållshanteringsmenyn

I menyn finns länkar för menyredigering, innehållsredigering, sidfotredigering, ”ansök nu”-knappen, användaradministration och för utloggning.

6.5.3 Menyhantering

Menyredigeringslänken i innehållshanteringsmenyn anropar första meny-nivåns kontroller-klass ”SubmenuController” och indexAction()-metod.

```
public function indexAction() {
      $menu = $this->_request->getParam ( 'id' );
      $mdlMenu = new Application_Model_DbTable_Menu ();
      $mdlMenu->setLang();
      $mdlSubMenu = new Application_Model_DbTable_SubMenu ();
      $mdlSubMenu->setLang();
      $this->view->submenu = $menu;
      $this->view->layout ()->page = $menu;
      $this->view->layout ()->id = $menu;
      $this->view->items = $mdlSubMenu->getItemsByMenu ( $menu );
}
```

Funktionen sköter om att söka upp alla första nivåns menyobjekt från databasen. Menyobjekten väljs från databasen med hjälp av huvudkategorins id som användaren är på. Menyobjekten visas för användaren i en tabell, där varje objekt har sin egen rad med länkar för att uppdatera, flytta, radera eller hantera undermenyobjekt. Varje länk representerar en Action-metod i kontrollerklassen, utom ”Hantera undermenyer”-länken som anropar kontroller-klassen för nästa nivå, som består av en identisk vy av menyobjekt.

I Figur 16 visas vyn för menyhanteringen.

Figur 16 Menyhantering

Ett nytt menyobjekt skapas via länken i menyhanterings-vyn, som anropar createAction()-metoden i respektive kontroller-klass.

```
public function createAction()
{
      $menu = $this->_request->getParam ( 'id' );
      $mdlMenu = new Application_Model_DbTable_Menu ();
      $mdlMenu->setLang();

      $frmSubMenu = new Application_Form_Submenu ();

if ($this->_request->isPost ()) {
            if ($frmSubMenu->isValid ( $_POST )) {
                  $data = $frmSubMenu->getValues ();
                  $mdlSubMenu = new Application_Model_DbTable_SubMenu
();
                  $mdlSubMenu->setLang();
                  if ($frmSubMenu->getElement ( 'checkbox' )-
>isChecked () && $data['page_id'] == 0) {

                        $link = null;
                        $pageNum = $data ['page_id'];
                        }
                        else if($data['page_id'] != 0)
                        {
                              $link = null;
                              $pageNum = $data ['page_id'];
                        }
                        else
{
                              $itemPage = new CMS_Content_Item_Page
();
$itemPage->name = $frmSubMenu->getValue ( 'label' );
                              $itemPage->headline = $frmSubMenu-
>getValue ( 'label' );
                              $itemPage->content = '';
                              $itemPage->parent_id = '';
                              $itemPage->menuid = $this->_request-
>getParam ( 'id' );
                              // save the content item
                              $pageNum = $itemPage->save ();
```

```
                        $link = '/page/' . $pageNum;

                    }

                    $mdlSubMenu->addItem ( $this->_request-
>getParam ( 'id' ), $data ['label'], $pageNum, $link );

                    $page = new CMS_Content_Item_Page ( $this-
>_request->getParam ( 'id' ) );
                    $array [] = $page->toArray ();

                    $this->view->layout ()->page = $array [0]
['menuid'];
                    $this->_request->setParam ( 'menu', $this-
>_request->getParam ( 'id' ) );
                    $this->_forward ( 'index' );
                }
}
```

Metoden sköter om att anropa Submenu-formuläret som visas för användaren, samt valideringen av data och lagring av data i databasen. Formuläret (se Figur 17) består av tre olika steg. Första steget är att välja beskrivning till menyobjektet. I andra steget väljs vart menyobjektet skall länka. Ifall valet är ”None”, skapar systemet en ny rad i databastabellen för innehållet genom att anropa createAction()-metoden i PageController-klassen.

```
public function createAction() {
            $menuid = $this->_request->getParam ( 'base' );
            $itemPage = new CMS_Content_Item_Page ();
            $itemPage->name =  '';
            $itemPage->headline = '';
            $itemPage->content = '' ;
            $itemPage->menuid = $menuid;
            // save the content item
            $itemPage->save ();
      }
```

Metoden sköter om att skapa länkningen med menyobjektet i databasen. Vid detta skede sparas ännu inget innehåll för själva sidan, utan detta kan användaren sedan skapa i ett senare skede (se kapitel 6.5.4). ”Ingen länk”-rutan hindrar systemet från att skapa en länk till menyobjektet för att kunna skapa undermenyer till den.

Figur 17 Formulär för att skapa menyobjekt

6.5.4 Redigera innehåll

Redigeringen av innehållet sker genom att navigera till en sida på webbplatsen och att trycka på ”Editera innehåll”-knappen i innehållshanteringsmenyn (se kapitel 6.5.2). Länken anropar editAction()-metoden i PageController-klassen.

```
public function editAction()
{
      $id = $this->_request->getParam ( 'id' );
      $itemPage = new CMS_Content_Item_Page ( $id );

      $pageForm = new Application_Form_PageForm ();
      $pageForm->setAction ( '/page/edit/'.$id );

if ($this->getRequest ()->isPost ())
{
            if ($pageForm->isValid ( $_POST ))
{

                  $itemPage->name = $pageForm->getValue ( 'name' );
                  $itemPage->headline = $pageForm->getValue ( 'head-
line' );
                  $itemPage->content = stripslashes($pageForm-
>getValue ( 'content' ));
                  // save the content item
                  $itemPage->save ();
                  $this->view->layout()->edit = false;
                  $this->view->layout()->$id;
                  return $this->_forward ( 'index' );
            }
      }
      $array[] = $itemPage->toArray();

$pageForm->populate ($itemPage->toArray());

      $this->view->layout()->page = $array[0]['menuid'];
      $this->view->layout()->id = $id;
      $this->view->form = $pageForm;
```

```
}
```

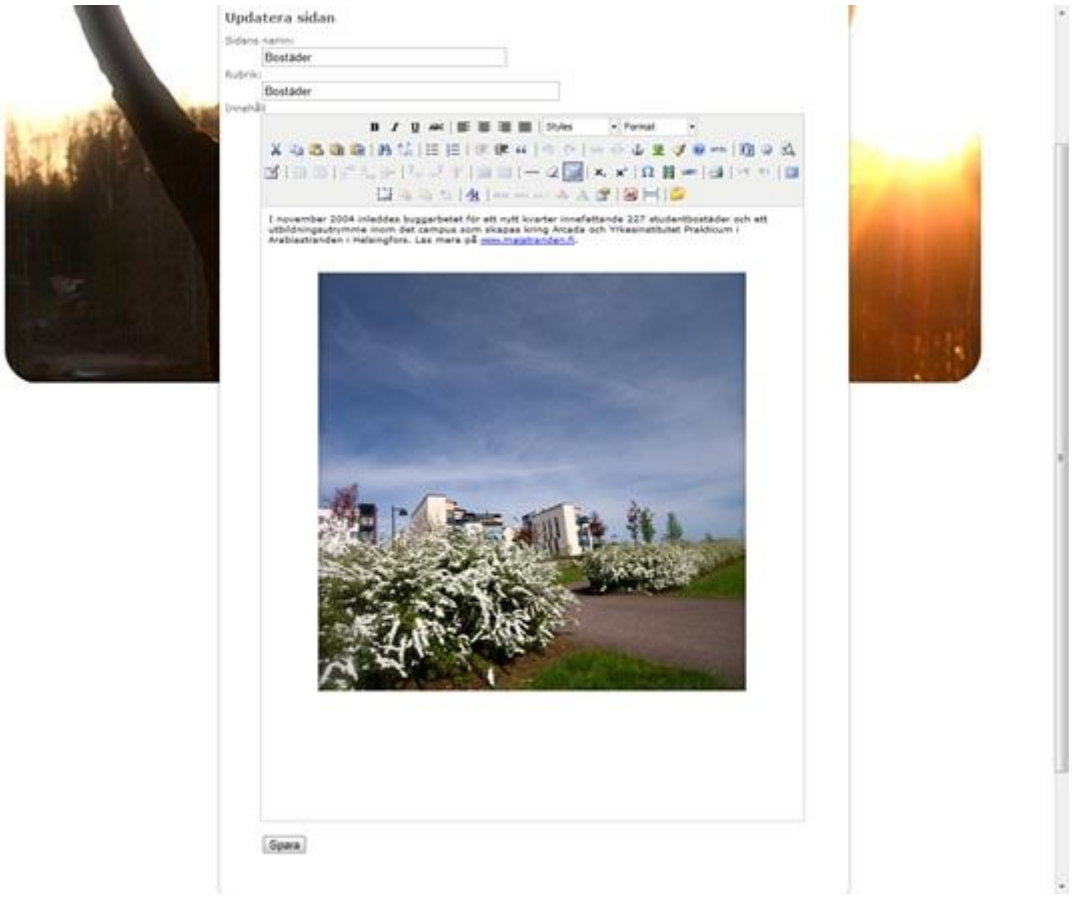
Metodens uppgift är att visa innehållet i ett redigeringsformulär för sidan. Detta sker genom att metoden anropar CMS_Content_Item_Page-klassen med en id vilken används för att hitta rätt innehåll från databasen med hjälp av Page-modellklassen. CMS_Content_Item_Page sparas sedan som ett objekt för att fylla i PageForm-formuläret med innehåll.

```
class Application_Form_PageForm extends Zend_Form {

      public function init() {
            $this->setAttrib ( 'enctype', 'multipart/form-data' );
            // create new element
            $id = $this->createElement ( 'hidden', 'id' );
            // element options
            $id->setDecorators ( array ('ViewHelper' ) );
            // add the element to the form
            $this->addElement ( $id );
            // create new element
            $name = $this->createElement ( 'text', 'name' );
            // element options
            $name->setLabel ( 'Sidans namn: ' );
            $name->setRequired ( TRUE );
            $name->setAttrib ( 'size', 40 );
            // add the element to the form
            $this->addElement ( $name );
            // create new element
            $headline = $this->createElement ( 'text', 'headline' );
            // element options
            $headline->setLabel ( 'Rubrik: ' );
            $headline->setRequired ( TRUE );
            $headline->setAttrib ( 'size', 50 );
            // add the element to the form
            $this->addElement ( $headline );
            // create new element
            $content = $this->createElement ( 'textarea', 'content' );
            // element options
            $content->setLabel ( 'Innehåll' );
            $content->setRequired ( TRUE );
            $content->setOptions(array('class' => 'content'));
            $this->addElement ( $content );

            $submit = $this->addElement ( 'submit', 'submit', array
('label' => 'Spara' ) );
      }

}
```

PageForm-formuläret öppnar TinyMCE-editorn, som används för att skapa innehåll i ett innehållshanteringssystem (se kapitel 5.2.1). Innehållet sparas i databasen i HTML-form

som skiljer sig en aning från det vanliga sättet att spara data i ett innehållshanteringssystem (se kapitel 5.2.2). Editorn innehåller också ett skräddarsytt insticksprogram med vilken man kan lägga till olika filmer som spelas upp med FlowPlayer. Dessa filmer spelas upp från Arcadas streaming-server.

I Figur 18 visas innehållshanteringen med TinyMCE-editorn för en av underkategorierna för ”Att studera på Arcada”.



Figur 18 Film & Tv – Innehållsredigering

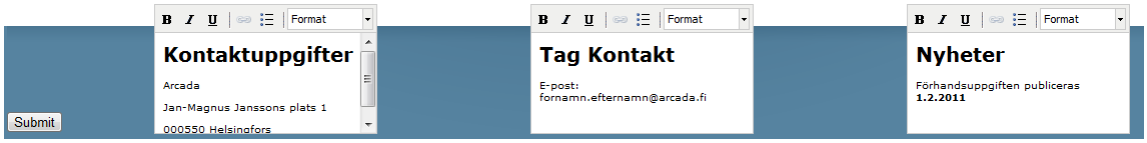
6.5.5 Redigera sidfoten

Redigeringen av sidfoten fungerar enligt samma princip som redigering av innehållet. När användaren klickar på ”Editera footern”– länken, visas TinyMCE-editor i varje textruta med innehåll och när redigeringen är klar, sparas innehållet i databasen. Skillnader jämfört med innehållsredigeringen är att tre separata editorer används för att enklare kunna hålla innehållet i sär. Skillnader finns också i sättet hur FooterController-klassens indexAction() och editAction()-metoder blir anropade. När resten av länkarna i innehållshanteringsmenyn anropar sina egna kontrollerklasser, kallar ”Editera footern”–länken på indexAction()-metoden med en parameter som indikerar om sidfoten skall vara i editerings läge eller inte. Ifall editerings-vyn är vald, anropas editAction-metoden från FooterController-klassen som sköter om att hämta data från databasen med Application_Model_DbTable_Footer-klassen och att sätta in den i Application_Form_Footer-formuläret.

```
class Application_Form_Footer extends Zend_Form {

      public function init() {
            $submit = $this->addElement ( 'submit', 'submit2', array
('label' => 'Submit' ) );
            $contactinfo = $this->createElement ( 'textarea', 'con-
tactinfo' );
            // element options
            $contactinfo->setOptions ( array ('class' => 'footer' ) );
            $contactinfo->removeDecorator('label');
            $this->addElement ( $contactinfo );

            $contact = $this->createElement ( 'textarea', 'contact' );
            $contact->removeDecorator('label');
            // element options
            $contact->setOptions ( array ('class' => 'footer' ) );
            $this->addElement ( $contact );

            $news = $this->createElement ( 'textarea', 'news' );
            $news->removeDecorator('label');
            // element options
            $news->setOptions ( array ('class' => 'footer' ) );
            $this->addElement ( $news );

      }

}
```

När användaren är färdig med redigeringen skall datat sparas. Detta sker med att trycka på ”Spara”-knappen som igen anropar editAction()-metoden. Denna gång vet funktionen att data skall sparas i databasen. Efter det sköter metoden om att adressera om användaren till framsidan och nollar parametern vilket berättar för systemet att indexAction()-metoden i FooterKontroller-klassen som skapar den vanliga vyn av sidfoten skall köras.

I Figur 19 visas redigerings-vyn för sidfoten med tre TinyMCE-editorer.



Figur 19 Film & Tv – Sidfot-redigering

6.5.6 Ansök nu

I Figur 20 visas en enkel konfigurationsformulärsida för ”Ansök nu”- knappen. Sidan består av ett formulär där man kan välja om ”Ansök nu”- knappen skall vara synlig eller inte och adressen till ”Ansök nu”-sidan. Formuläret Application_Form_Applynow anropas och fylls i från databasen med ApplynowController-klassens editAction()-metod, efter att användaren har tryckt på ”Ansök nu” – länken i innehållshanteringsmenyn.

Figur 20 Film & Tv – ”Ansök nu”-knappens redigering

7 DISKUSSION

Den färdiga produkten blev en välfungerande webbapplikation och ett kraftfullt men enkelt innehållshanteringssystem vilket uppfyllde förväntningarna beställaren hade. Alla utvecklade klasser fungerar bra ihop och efter en kort testperiod är webbplatsen nu publicerad.

7.1 Innehållshanteringssystem

Ett av målen för examensarbetet var att analysera vilka egenskaper och vilken funktionalitet krävs av ett innehållshanteringssystem för att det skall vara bra och fungerande. Innehållshanteringssystem finns av många olika slag och kan användas till olika typer av webbapplikationer. Men för att kallas användbart och bli accepterat, måste det vara möjligt att skapa, hantera, spara och presentera innehåll i ett enkelt sätt. Dessutom krävs att innehållshanteringssystem skall kunna hantera hundratusentals användare på ett säkert sätt.

Ett annat mål var att jämföra skräddarsydda innehållshanteringssystem med proprietära system och system med öppen källkod. Skräddarsydda innehållshanteringssystem har sina fördelar i optimerad kod och funktionalitet anpassad för webbapplikationen. I proprietära system och system med öppen källkod blir man ofta tvungen att göra svåra lösningar med funktioner som är inte avsedda för ändamålet. Sådana lösningar kan göra hela applikationen långsam och administrationen krånglig. Nackdelarna jämfört med innehållshanteringssystem som är proprietära eller har öppen källkod är att det alltid behövs programmerare för att sätta upp ett skräddarsytt system och för att hålla systemet uppdaterat och säkert. Skräddarsydda system är oftast också inkompatibla med andra webbapplikationer.

Kostnaderna att skapa ett skräddarsytt innehållshanteringssystem blir ändå ofta större jämfört med en ”öppen källkods”-produkt och kan till och med bli större än för proprietära produkter. Orsaken är att ”öppen källkods”-produkter är gratis och proprietära produkter har oftast ett rimligt pris. Ett skräddarsytt innehållshanteringssystem behöver alltid en eller flera programmerare som skapar och administrerar produkten och detta kan bli dyrt.

7.2 Zend Framework

Ett tredje mål med examensarbetet var att lära sig hur PHP-ramverk, t.ex. Zend Framewrok, fungerar och analysera nyttan med användning av ramverk. Zend Frameworks enorma mängd färdiga funktioner och moduler möjliggör en mycket snabb utvecklingsprocess. Samtidig innebär det också en brant inlärningskurva, speciellt om man inte har tidigare erfarenhet av liknande system. För att kunna utnyttja ramverket maximalt måste man förstå ramverkets arkitektur och framför allt, hur det är tänkt att funktionalitet skall implementeras. Utan denna kunskap skapas tyvärr ofta onödig och komplicerad kod vilket kan orsaka mer problem än vad det egentligen löser. Dessutom kan säkerhetsluckor uppstå om utvecklaren inte känner till hur autentisering och hantering av användare fungerar med ramverket.

En praktisk målsättning för arbetet var att klarlägga hur klasser och funktioner utvecklas med hjälp av Zend Framework. Klasserna struktureras enligt MVC-modellen i Zend Framework och därför är det lätt att ombryta koden i olika klasser beroende på deras uppgift. Exempel på klasser är vy, kod för visning av helheten, kod för logiken till modellen och kod som sköter om att styra systemet enligt förfrågningarna till kontrollern. Varje förfrågning handskas som en operation i kontrollern och är implementerad som en metod i koden. Genom att implementera en Zend_Layout kan man på ett behändigt sett skapa en enhetligt utseende för varje sida i webbplatsen. I Kapitel 6 visas med hjälp av kod från projektet hur man strukturerat funktionerna i verkligheten.
En annan praktisk målsättning var at ta reda på hur man åsidosätter standardfunktionalitet i Zend Framework, vilket visade sig vara mycket enkelt. Många klasser i ramverket har en disable()-funktion för att tala om för systemet att en särskild funktionalitet icke skall köras. Detta används till exempel för framsidans banner där Zend_Layout åsidosätts för att komma åt datavariabler genom att anropa en Zend kontroller-klass.

7.2.1 Fördelar

Zend Framework har en hel del fördelar som gör att ramverket ofta används. Systemet innehåller många färdiga och användbara moduler, som möjliggör en snabb utvecklingsprocess. Arkitekturen tillåter användning av flera vyer och sub-vyer som gör det lätt att skapa dynamiska sidor med enhetligt utseende och hjälper till att hålla koden organiserad.

En stor fördel med Zend Framework är dess åtkomstkontroll som är inbyggd i ramverket. Genom att implementera en Zend åtkomstkontroll-lista kan man lätt bestämma vilka användarroller har åtkomst till vilka operationsfunktioner. Detta möjliggör att man kan enkelt hantera rättigheter på metodnivå.

7.2.2 Nackdelar

Zend Frameworks egna klasser är mycket svårlästa på grund av hög abstraktion vilket gör det svårt att reda ut hur något fungerar i ramverket. För att förstå logiken bakom en specifik funktion kan man bli tvungen att bläddra genom ramverksfiler utan att hitta koden. En annan nackdel med Zend är att systemet är ämnat för en ganska bred användargrupp. Detta innebär att det finns en hel del färdiga moduler och lösningar i ramverket som kan vara svåra att anpassa enligt egna behov fastän dokumentationen för ramverket är mycket välgjord.

7.3 Slutsatser

Det är uppenbart att Zend Framework är ett mycket lämpligt ramverk för utveckling av webbapplikationer. Men efter att jobbat med det, är min erfarenhet den att även enklare ramverk med färre färdiga moduler och funktioner, skulle räcka till. Inlärningstiden skulle då avsevärt förkortas. För vissa funktioner var jag tvungen att använda mig av en egen lösning på grund av att jag inte riktigt förstod hur ramverkets modul fungerade och om det skulle förkorta utvecklingstiden. Ett exempel på en sådan funktion är implementation av flera språk som nu löstes med olika databastabeller för varje språk. Uppenbarligen skulle en implementation av Zend_Translate och en väldesignad databastabell ha varit en elegantare lösning men skulle ha krävt en hel del mera

utvecklingstid. Andra förbättringar skulle vara att ta cache-funktionalitet i bruk och att använda mer optimerade databassatser. Dessa skulle göra webbplatsen ännu snabbare.

Att utveckla ett skräddarsytt innehållshanteringssystem var ett bra val för mitt examensarbete, inte bara för att beställaren tycker att webbplatsen är mycket enkel att använda och underhålla utan också för att kunskapen jag har fått är mycket större än vad man skulle ha fått i implementering av ett färdigt system. Dessutom är innehållshanteringssystemet optimerat och välplanerat för att fungera med denna webbapplikation. Uppenbart kunde också ett proprietärt eller ”öppen källkods”-innehållshanteringssystem ha använts. Men på grund av webbplatsens speciella design, skulle implementeringen av ett färdigt system ha blivit minst lika tidskrävande på grund av tilläggsmoduler som i alla fall skulle programmeras.

KÄLLOR

Allen & Lo & Brown. 2007. Zend Framework in Action. Manning Publication

ISBN: 978-1933988320

Brampton, Martin. 2008. PHP5 CMS Framework Development. Packt Publishing.
ISBN: 978-1-847193-57-5

Lyman, Forrest. 2009. Pro Framework Techniques. Build a full CMS Project.
ISBN: 978-1-4302-1879-1

Pope, Keith 2009 Zend Framework 1.8 Web Application Development
ISBN: 978-1-847194-22-0

Peacock, Michael. 2009. Drupal 6 Social Networking. Packt Publishing.
ISBN: 978-1-847196-10-1

Shan, Tony 2006. Taxonomy of Java Web Application Frameworks. Proceedings of 2006 IEEE International Conference on e-Business Engineering.
ISBN:0-7695-2645-4

Freeman, Eric & Elisabeth. 2004. Head First Design Patterns.O'Reilly

Programmer's Reference Guide: Zend Framework 2010a, Zend Controller , [www] Hämtat 20.1.2011 http://framework.zend.com/manual/en/zend.controller.html

Programmer's Reference Guide: Zend Framework 2010b, Zend Controller , [www] Hämtat 22.1.2011 http://framework.zend.com/manual/en/zend.acl.html

Programmer's Reference Guide: Zend Framework 2010c, Zend Controller , [www] Hämtat 24.1.2011 http://framework.zend.com/manual/en/zend.auth.html

Programmer's Reference Guide: Zend Framework 2010d, Zend Controller , [www] Hämtat 24.1.2011 http://framework.zend.com/manual/en/zend.translate.html

Programmer's Reference Guide: Zend Framework 2010e, Zend Controller , [www] Hämtat 25.1.2011 http://framework.zend.com/manual/1.11/en/zend.json.html

Programmer's Reference Guide: Zend Framework 2010f, Zend Controller , [www] Hämtat 29.1.2011 http://framework.zend.com/manual/1.11/en/zend.service.html

Programmer's Reference Guide: Zend Framework 2010g, Zend Controller , [www] Hämtat 30.1.2011
http://framework.zend.com/manual/1.11/en/zend.application.core-functionality.html

Programmer's Reference Guide: Zend Framework 2010h, Zend Controller , [www]
Hämtat 15.2.2011
http://framework.zend.com/manual/en/zend.controller.actionhelpers.html

Techchorus 2009, Reasons to use zend framework – [www]
Hämtat 29.1.2011. Senast modifierat 7.12.2009, 18:06.
http://techchorus.net/reasons-use-zend-framework

Docforge 2011, Web application framework
[www]. Hämtat 04.05.2011. Senast modifierat 27.6.2010, 14:59.
http://docforge.com/wiki/Web_application_framework

Riehle, Dirk 2000, Framework Design: A Role Modeling Approach
[www]. Hämtat 04.05.2011. Senast modifierat 2000.
http://dirkriehle.com/computer-science/research/dissertation/diss-a4.pdf

Wikipedia 2011, Ohjelmistokehys – Wikipedia, the free encyclopedia
[www]. Hämtat 29.1.2011. Senast modifierat 14.02.2011, 04:06 .
http://fi.wikipedia.org/wiki/ Ohjelmistokehys

Webaxes 2010, Why Zend Framework
[www]. Hämtat 04.05.2011. Senast modifierat 21.05.2010.
http://www.webaxes.com/2010/05/why-zend-framework/

Adobe 2005, ADOBE TO ACQUIRE MACROMEDIA
[www]. Hämtat 08.5.2011. Senast modifierat 18.04.20005.
http://www.adobe.com/macromedia/proom/pr/2005/adobe_macromedia.html

Object orientation tips 1998, Model-View-Controller
[www]. Hämtat 15.3.2011. Senast modifierat 14.4.1998.
http://ootips.org/mvc-pattern.html

BILAGA 1: JÄMFÖRELSE AV PHP-RAMVERK

PHP

[edit]

Project	Language	Ajax	MVC framework	MVC Push/Pull	i18n & l10n?	ORM	Testing framework(s)	DB migration framework(s)	Security Framework(s)	Template Framework(s)	Caching Framework(s)	Form Validation Framework(s)
CakePHP	PHP	Prototype/script.aculo.us, jQuery/jQuery UI, MooTools/MooTools more	Yes	Push	Yes	Active record pattern (CakePHP 1.x), Data Mapper Pattern (CakePHP 2.x)	Unit Tests, Object Mocking, Fixtures, Code Coverage, Memory Analysis with SimpleTest and XDebug	Yes	ACL-based	Themes, Layouts, Views and Elements	Memcache, XCache, APC, File	Validation and Security
CodeIgniter	PHP >= 5.1	Prototype/script.aculo.us, jQuery/jQuery UI	Modified active record pattern	Push	Plugins	No	Unit Tests	No	Yes	Yes	Yes	Yes
Drupal	PHP	jQuery/jQuery UI, more	No	Push & Pull	Yes	Optional module	SimpleTest	Yes	Yes	Yes	Memcache, APC, Varnish and more	Yes
eZ Components	PHP	No	Yes	Push	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Fusebox	PHP	Yes	Not mandatory	Push	No, custom				Multiple plugins available			via qforms or built in PHP validation
Joomla	PHP >= 4.4.x	No	Yes	Push & Pull	Yes	Yes	Unit Tests	No	Yes	Yes	Yes	No
Kajona	PHP 5	Yes	Yes	Yes	Yes	Yes	Unit Tests	Automatic	Yes	Yes	Yes	Yes
Kohana	PHP 5	Yes	MVC, HMVC	Push	Yes	Yes	PHPUnit, (as module)	Yes	Plugin	Yes	APC, Database, eAccelerator, File, Memcache, XCache	Yes
Midgard	PHP	jQuery	MidCOM	Pull	Yes	Midgard	PHPUnit	datagard	ACL-based	PHP and TAL	Memcache	Yes
Qcodo	PHP 5	built-in	QControl	Push	Yes	XML-based		Inherent		QForm and QControl	Yes	Yes
RedSpark	PHP 5	jQuery	Yes	Push & Pull	Yes	Yes	Yes	Yes	ACL-based	Yes	Yes	Yes
Symfony	PHP 5	Prototype, script.aculo.us, Unobtrusive Ajax with UJS and PJS plugins	Yes	Push	Yes	Propel, Doctrine (YAML)	Yes	Plugin exists (alpha code)	Plugin	Yes	Yes	Yes
Yii	PHP 5 (>=5.1.0)	jQuery, jQuery UI, own components, plugins	Yes	Push & Pull	Yes	Database Access Objects (DAO), Active Record, plugins	PHPUnit, Selenium	Yes	ACL-based, RBAC-based, plugins	PHP-based, PRADO-like, plugins	APC, Database, eAccelerator, File, Memcache, WinCache, XCache, Zend Platform	Yes
Zend Framework	PHP 5 (>=5.2.4)	Toolkit-independent	Yes	Push & Pull	Yes	Table and Row data gateway	Unit Tests	Yes	ACL-based	Yes	Yes	Yes
SilverStripe (Sapphire)	PHP 5 (>=5.2)	jQuery/jQuery UI	Active record pattern	Push & Pull	Yes	Active record pattern	Unit Tests	(Automatic)	incl. OpenID	(object oriented)	Yes	Yes
Project	Language	Ajax	MVC framework	MVC Push/Pull	i18n & l10n?	ORM	Testing framework(s)	DB migration framework(s)	Security Framework(s)	Template Framework(s)	Caching Framework(s)	Form Validation Framework(s)